User's Manual

Renesas Synergy™ Software Package (SSP) V2.7.0

User's Manual

Renesas Synergy™ Platform

Renesas Electronics
www.renesas.com

Revision 1.70 Mar.18.2025

Table of Contents

Chapter 1 Renesas Synergy™ Software Package Introduction

1.1 Introduction to the SSP User's Manual

This manual describes how to use the Renesas Synergy Software Package for writing applications for the Synergy microcontroller series. In the figure below, the API Reference components of the SSP User's Manual are indicated in blue. Additional components such as the description of the e2 studio ISDE and tutorials and example applications are included in this manual to guide you through the steps of programming with the SSP.

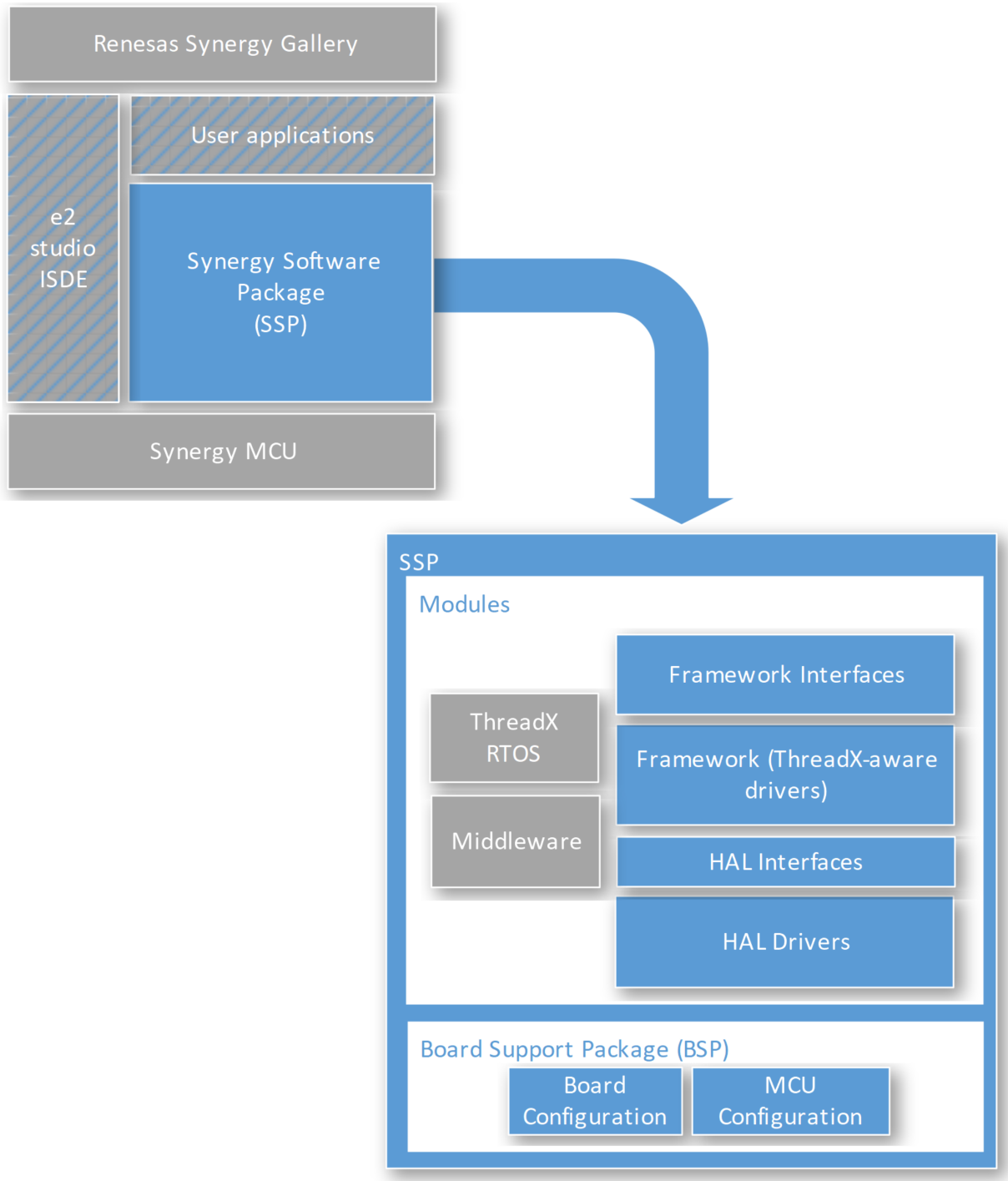


Figure 1: Synergy Software Package (SSP) Documentation

1.2 Subjects Covered in this Manual

To learn about the SSP architecture and about board and chip-level support included in the SSP see:

- SSP Architecture
- BSP Architecture

For programming with the SSP and an introduction to the e2 studio ISDE see:

e2 studio ISDE User Guide

For introductory tutorials and application examples see:

- Tutorial Blinky
- Application WDT

For Module Overviews describing the SSP Modules, see:

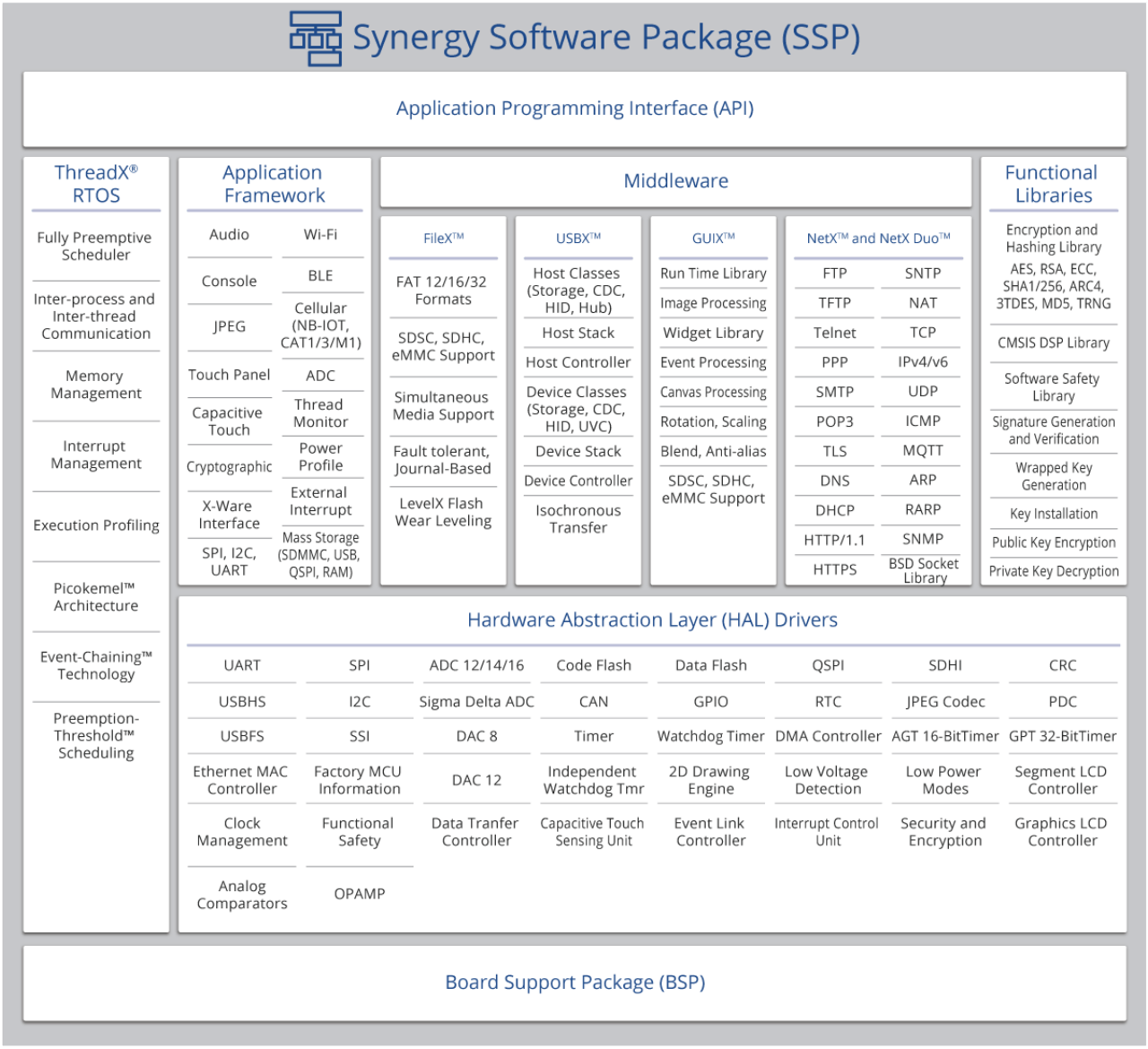
- Framework Layer
- HAL Layer

The API reference documentation of the following SSP components is included in this document:

- Framework Interfaces for Interfaces to the ThreadX-aware Framework Modules
- Framework Layer for ThreadX-aware Framework Driver Modules
- HAL Interfaces for Interfaces to the Hardware Abstraction Layer (HAL) Modules
- HAL Layer for the RTOS-independent HAL driver Modules
- Board Support Package for the Board Support Package (BSP) which includes board-specific and microcontroller-specific configuration modules

Chapter 2 SSP Overview

2.1 SSP Overview



Learn how to develop applications with the Synergy Software Package (SSP) using the SSP's module-based architecture and the functional software layers. Integrate SSP applications with multiple boards and Synergy devices using the Board Support Package (BSP).

The following pages describe the fundamental SSP architecture:

- SSP Architecture
- BSP Architecture

2.1.1 Introduction

2.1.1.1 Purpose

The Renesas Synergy™ Software Package (SSP), part of the Renesas Synergy™ Platform, is a complete integrated software package designed to provide easy to use, scalable, high quality software for embedded system design. Using the Synergy Software Platform will reduce time to market by providing a completely integrated and qualified embedded software platform comprising of an industry leading, completely optimized and hardened Real-time Operating System (RTOS), middleware, communication stacks, function libraries, application framework and hardware abstracted low-level device drivers.

2.1.1.2 Overview

The SSP is divided into four main layers:

- Framework Interfaces The Framework Library connects to Synergy hardware peripherals through common Interfaces, which abstract the hardware into functional use cases. The Interface layer is a group of header files with definitions of functions and parameters, so it consumes no code space.
- Framework Layer The Framework layer consists of RTOS integrated drivers and valuable application code.
- HAL Interfaces HAL layer Interfaces connect to RTOS-independent HAL-level drivers.
- HAL Layer The HAL layer drivers with hardware registers implement Interfaces.

2.1.1.3 Ease of Use

The SSP provides uniform and intuitive APIs that are well documented. Each module is supported with detailed user documentation and software datasheet including code size and execution time for each function.

2.1.1.4 Scalability

Users have the choice to integrate with the platform capabilities using either the Framework Interface or HAL layer, depending on which best meets the needs of the application. To further scale each module, build time options such as parameter checking may be compiled out for smaller, more efficient code.

2.1.2 SSP Architecture

2.1.2.1 Renesas Synergy Software Package (SSP) Architecture

This section describes the Renesas Synergy Software Package (SSP) architecture and how to use the SSP Application Programming Interface (API).

Introduction to the SSP

As microcontrollers increase in complexity, so does the breadth of knowledge required to make them

operate in the desired way. The SSP provides an innovative approach to embedded software for IoT applications. With the SSP, you have a new and extremely powerful software interface from the ground up, making coding fast and providing you with a robust development processes. With this software, you can create differentiated application code instead of spending months developing baseline code to interface at the hardware level.

SSP Terms

Term	Description	Reference
Module	Modules can be peripheral drivers, purely software, or anything in between. Each Module consists of a folder with source code, documentation, and anything else that the customer needs to use the code effectively. Modules are independent units, but they may depend on other Modules. Example SSP Modules are the UART driver (UART Interface), Audio Playback Framework, which relies on timer, DMA, and DAC drivers (Audio Framework Interface), or the Messaging Framework (Messaging Framework Interface). Applications can be built by combining multiple modules to provide the user with the features they need.	SSP Modules
BSP	Short for Board Support Package. In the SSP the BSP provides just enough foundation to allow other SSP modules to work together without issue.	BSP Architecture

Callback Function	This term refers to a function that is called when an event occurs. For example, the bus error interrupt handler is implemented in the r_bsp. The user will likely want to know when a bus error occurs. To alert the user, a callback function can be supplied to the r_bsp. When a bus error occurs the r_bsp will jump to the provided callback function and the user can handle the error. Interrupt callback functions should be kept short and be handled carefully because when they are called the MCU will still be inside of an interrupt and therefore will be delaying any pending interrupts.	-
Interface	See SSP Interfaces section: SSP Interfaces. All interfaces in the SSP are listed here: Framework Interfaces and HAL Interfaces	SSP Interfaces
Instance	See SSP Instances section: SSP Instances	SSP Instances
Module Instance	Single and independent configuration of a Module.	-
Application	Code that is owned and maintained by the user. Application code may be based off sample application code provided by Renesas, but is the responsibility of the user.	An example for a simple application is included as tutorial in this manual: ref application-wdt
Driver	A Driver is a specific kind of Module that directly modifies registers on the MCU.	-
Stacks	The SSP architecture is designed such that Modules work together to form a Stack. Starting with the uppermost Module and going to the bottommost dependency forms a specific Stack.	SSP Stacks

Layer/Level	Stacks are made of multiple layers of Modules. A Layer can consist of one or multiple Modules depending on the requirements of the next Layer up. Layer and Level are used interchangeably.	ref ssp-predefined-layers

2.1.2.2 SSP Modules

Modules are the core building block of SSP. Modules can do many different things, but all Modules share the basic concept of providing functionality upwards and requiring functionality from below.

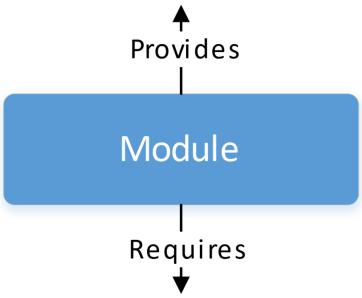


Figure 2: Modules

The amount of functionality provided by a Module is not limited though there are usually points where separation makes sense. If too much functionality is provided, then reuse of the Module can become difficult in the future. If not enough functionality is provided, then unnecessary complexity and overhead may be added in order to make the Modules work as expected.

The simplest SSP application consists of one Module with the user application on top.

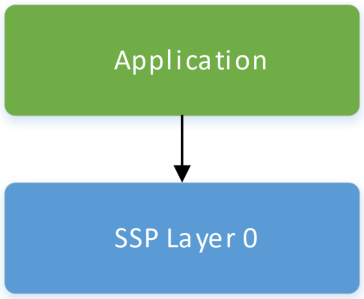


Figure 3: Module with application

For simplicity, ignore the Board Support Package (BSP) for now because it is a requirement of any

SSP project. In the picture above, the BSP is located underneath the bottom layer, SSP Layer 0.

2.1.2.3 SSP Stacks

When modules are layered atop one another, an SSP stack is formed. The stacking process is performed by matching what one module provides with what another module requires. For example, the Audio Playback Framework Module requires a Transfer Interface, which can be fulfilled by the Data Transfer Controller (DTC) Driver Module. Instead of including the DTC code in the Audio Playback Module, we split these into two modules. This allows for reuse of the underlying modules, which has many benefits.

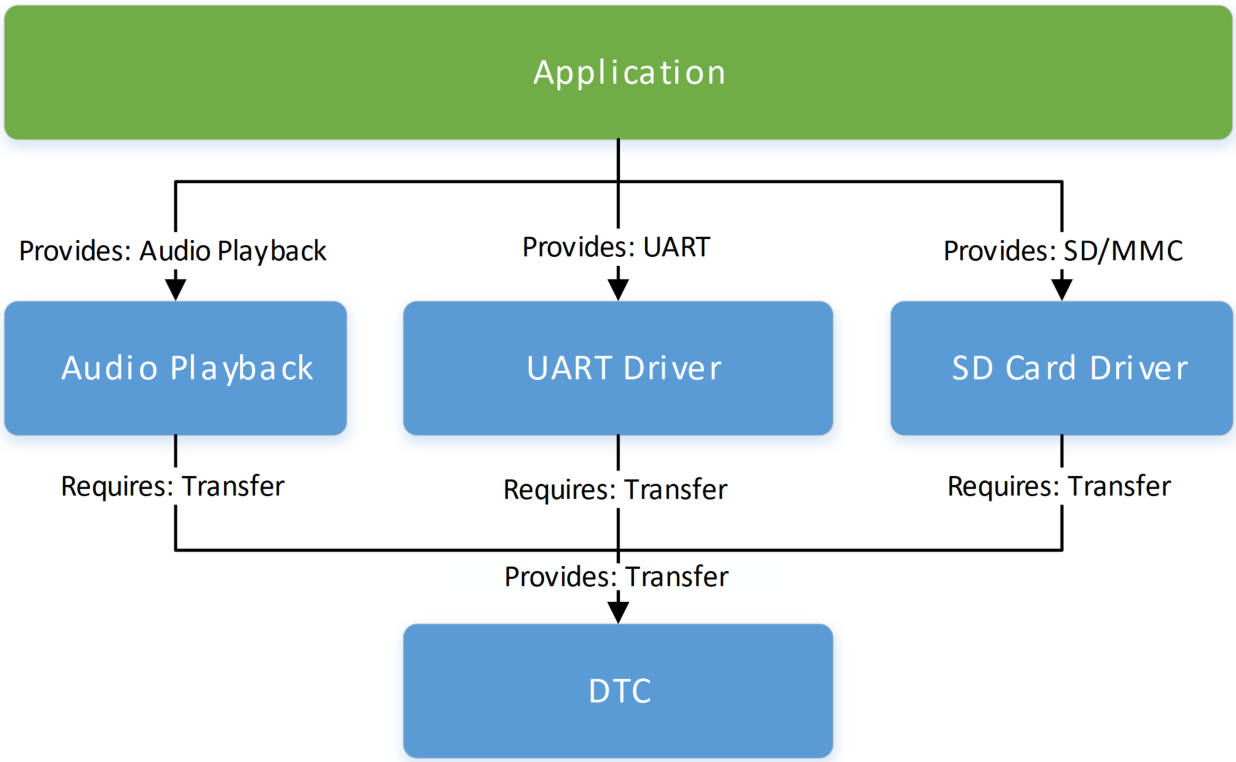


Figure 4: Stacks – Audio playback

By continuing to add layers to the Stack using SSP Modules, you can interface with the Synergy hardware at a high level.

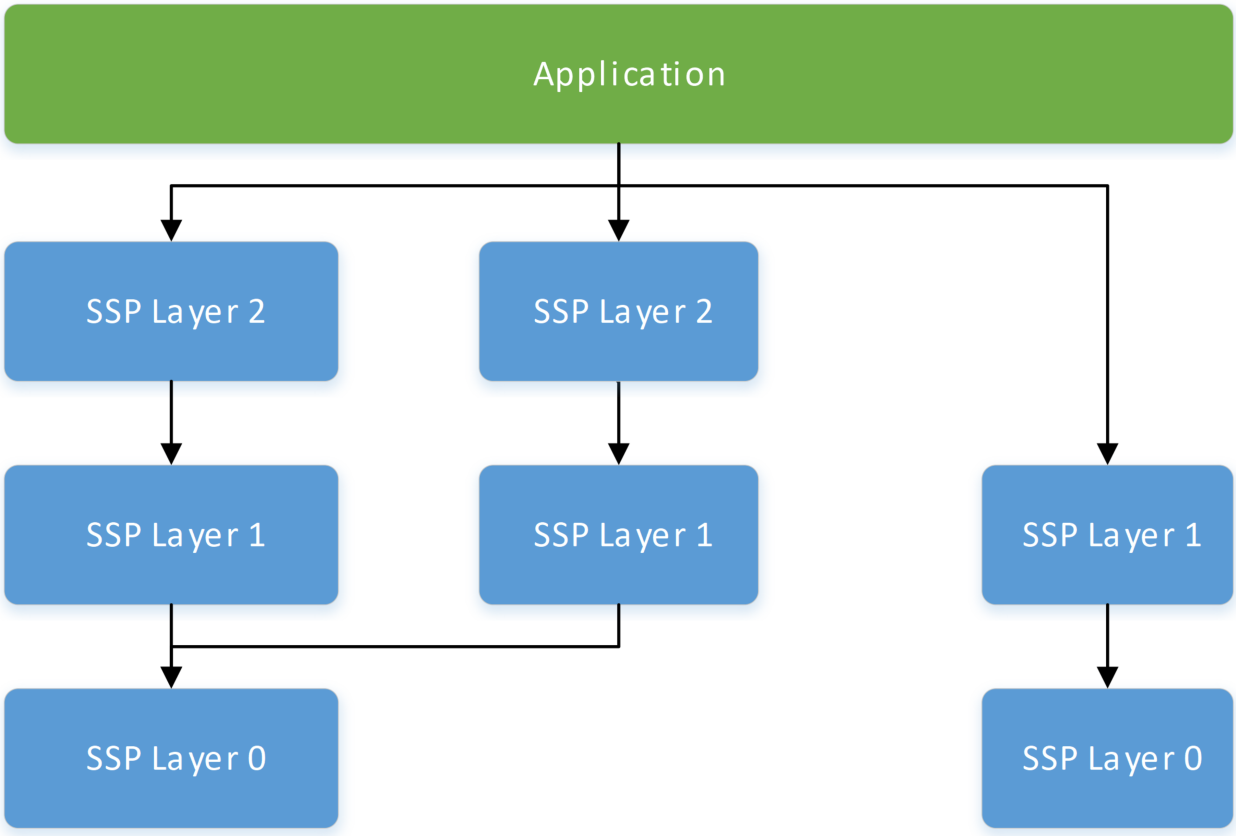


Figure 5: Stacks

The ability to stack modules has great benefit because it ensures the flexibility of the architecture as a whole. If modules are directly dependent upon other modules, then issues arise when application features must work across different user designs. To ensure that modules are reusable the modules must be capable of being swapped out for other modules that provide the same features. The SSP architecture provides this flexibility to swap modules in and out through the use of SSP Interfaces.

2.1.2.4 SSP Interfaces

At the architecture level, Interfaces are the way that Modules provide common features. This commonality allows Modules that adhere to the same Interface to be used interchangeably. Interfaces can be thought of as a contract between two Modules. The Modules agree to work together using the information that was agreed upon in the contract.

On Synergy hardware there is occasionally an overlap of features between different peripherals. For example, I2C communications can be achieved through use of the IIC peripheral or the SCI peripheral in its Simple I2C mode. There is a difference in the level of features provided by both peripherals. In I2C mode the SCI peripheral will only support a subset of the features of the full-featured IIC.

Interfaces aim to provide support for the common features that most users would expect. This means that some of the advanced features of a peripheral, such as the IIC, might not be available in

the Interface. In most cases these features are still available through Interface Extensions.

In design, Interfaces are implemented as header files. All Interface header filenames end with '_api.h'. The following sections detail what makes up an Interface.

SSP Interface Enumerations

Whenever possible, Interfaces use typed enumerations for function parameters and structure members.

```
typedef enum e_i2c_addr_mode
{
     I2C_ADDR_MODE_7BIT = 1, // Use 7-bit addressing mode
     I2C_ADDR_MODE_10BIT // Use 10-bit addressing mode
} i2c_addr_mode_t;
```

Enumerations remove any uncertainty when deciding what values are available for a parameter. Also note that enumeration options follow a strict naming convention where the name of the type is prefixed on the available options. Combining the naming convention with the autocomplete feature available in e^2 studio provides the benefits of rapid coding while maintaining highly readable code.

SSP Interface Data Structures

At a minimum, all SSP Interfaces include three data structures: a control structure, a configuration structure, and an instance structure.

The control structure is used as a unique identifier for using the module. If SSP modules were only peripheral drivers then this control structure might be replaced with a channel number. All function calls for that module would then take a channel number so that the code could determine which peripheral channel to operate on. SSP modules are not restricted to device drivers and therefore the control structure is used. The user allocates storage for a control structure and then sends a pointer to it into the open() call for a Module. At this point, the Module initializes the structure as needed. You must then send in a pointer to the control structure for all subsequent module calls. The contents of the control structure are used by the module and must not be altered. Reading data from a control structure should also be avoided as the data structure is not guaranteed to remain the same between SSP releases. In general, you should treat control structures like a black box.

The contents of a control structure are specific to an instance. This means that two instances of the same interface will have two completely different control structure types. The control structures that exist in an interface are named <interface>_ctrl_t. Below is an example of the interface control structure for I2C:

```
typedef void i2c_ctrl_t;
```

Since all interface control structures are of type void, they cannot be allocated. Instead, these types

are placeholders for instance control structures. Instance control structures are defined in instance header files and are named <instance>_instance_ctrl_t. Below are examples of I2C Instance control structures for the IIC (r_riic) and SCI (r_sci_i2c) peripherals:

```
/* riic Instance control structure to be used with I2C Interface. */
typedef struct st_riic_instance_ctrl
{
    i2c_cfg_t info; // Information describing I2C device
    uint32_t open; // Flag to determine if the device is open
    void * p_reg; // Base register for this channel

} riic_instance_ctrl_t;
/* sci_i2c Instance control structure to be used with I2C Interface. */
typedef struct st_sci_i2c_instance_ctrl
{
    i2c_cfg_t info; // Information describing I2C device
    uint32_t open; // Flag to determine if the device is open
    void * p_reg; // Base register for this channel
 /* More members specific to sci_i2c Instance. */
} sci_i2c_instance_ctrl_t;
```

When using an interface, the instance control structure should be allocated and used in place of the interface control structure. Using the example above, if the SCI-I2C Instance was being used then you would allocate a structure of type sci_i2c_instance_ctrl_t and use it wherever i2c_ctrl_t is referenced in the Interface. The ISDE will take care of allocating the correct control structure for you.

Dynamic memory allocation through use of the malloc() and free() functions are not used in SSP modules.

The configuration structure is used for the initial configuration of a module during the open() call. The structure consists of members such as: channel, interrupt priority, bitrate, and operating mode. The structure is used purely for input into the module. This structure does not have to be unique and could be discarded by you after initialization, if desired.

```
typedef struct st_i2c_cfg
{
    /* Generic configuration */
```

```
    uint8_t         channel;                                    // Identifier
recognizable by implementation
    i2c_rate_t      rate;                                       // Device's maximum
clock rate from enum i2c_rate_t
    uint16_t        slave;                                      // The address of the
slave device
    i2c_addr_mode_t addr_mode;                                  // Indicates how
slave fields should be interpreted
    uint16_t        sda_delay;                                  // The SDA output
delay
    uint8_t         rxi_ipl;                                    // Receive interrupt
priority
    uint8_t         txi_ipl;                                    // Transmit interrupt
priority
    uint8_t         tei_ipl;                                    // Transmit end
interrupt priority
    uint8_t         eri_ipl;                                    // Error interrupt
priority

    /* DTC/DMA support */
    transfer_instance_t const * p_transfer_tx;            // DTC instance for I2C
transmit.Set to NULL if unused.
    transfer_instance_t const * p_transfer_rx;            // DTC instance for I2C
receive. Set to NULL if unused.

    /* Parameters to control software behavior */
    void         (* p_callback)(i2c_callback_args_t * p_args);  // Pointer to
callback function
    void const    * p_context;                                  // Pointer to the
user-provided context

    /* Implementation-specific configuration */
    void const    * p_extend;                                   // Any configuration
data needed by the hardware
```

```
} i2c_cfg_t;
```

Above is an example configuration structure for the I2C Interface. The last three structure members (p_callback, p_context, and p_extend) are common to almost all module configurations.

The p_callback and p_context members are described in the SSP Interface Callback Functions section.

The p_extend member is used for extending the current Interface for a specific Instance. Interfaces are designed to support the most common features. There are cases where an Instance of an Interface requires extra information to properly configure itself. There are also cases where the extra information is not required, but users might need it to adjust the module for their specific application. When this is the case, the user can provide the underlying Instance with more configuration information by passing it through the p_extend member. The information that is passed through this member is defined by the underlying Instance, and therefore the user must adhere to its structure. If invalid information is passed to the underlying driver, then the Instance is not able to successfully use the data and proper operation cannot be guaranteed. Refer to the Interface Extensions section for more information.

It is also important that configuration structures only have members that apply to the current Interface. If multiple layers in the same stack define the same configuration parameters then it becomes difficult to know where to modify the option. For example, the baud rate for a UART is only be defined at the Driver layer. Any layers that use the UART Interface rely on the baud rate being provided at the Driver layer and do not offer it in their own configuration structures.

SSP Interface Callback Functions

Callback functions allow Modules to asynchronously alert the user application when an event has occurred. An example for an event is when a byte has been received over a UART channel. Callbacks are required to allow user application code to react to interrupts. SSP Modules define and handle the interrupts for Synergy MCU peripherals. If the user tries to define the interrupt service routine at the same time as a SSP Module, then the code does not build. Therefore SSP Modules allow the user application to respond to interrupts by registering a function to be called when an interrupt occurs.

Callback functions must be defined in the user application. They always return void and take a structure for their one parameter. The structure typedef is provided in the Interface for the Module and is named * _callback_args_t*. The contents of the structure may vary depending on the Interface, but two members are common: event and p_context.

The event member is used by the application to determine why the callback was called. Using the UART example again, the callback could have been triggered because a byte was received, all bytes had been transmitted, or a framing error has occurred. The event member is an enumeration provided by the Interface.

The p_context member is used for providing user-specified data to the callback function. In many cases a callback function is shared between multiple channels or Module Instances. When the callback occurs, the code handling the callback needs context information so that it can figure out which Module Instance the callback is for. For example, if the callback wanted to make a SSP API call in the callback, then at a minimum the callback must use the control structure. To make this easy, the user can provide a pointer to the control structure as the p_context. When the callback occurs,

the control structure is available as it will be passed in the callback structure.

Callback functions are called from within an interrupt service routine. For this reason callback functions should be kept as short as possible so they do not affect the real time performance of the user's system. An example skeleton function for the Flash Interface callback is shown below.

```
static void flash_callback (flash_callback_args_t * p_args)
{
    /* See what event caused this callback. */
 switch (p_args-&gt;event)
  {
        case FLASH_EVENT_ERASE_COMPLETE:
        /* Handle event. */
        break;
        case FLASH_EVENT_WRITE_COMPLETE:
        /* Handle event. */
        break;
        case FLASH_EVENT_BLANK:
        /* Handle event. */
        break;
        case FLASH_EVENT_NOT_BLANK:
        /* Handle event. */
        break;
        case FLASH_EVENT_ERR_DF_ACCESS:
        /* Handle error. */
        break;
        case FLASH_EVENT_ERR_CF_ACCESS:
        /* Handle error. */
        break;
        case FLASH_EVENT_ERR_CMD_LOCKED:
        /* Handle error. */
        break;
    }
}
```

When a Module is not directly used in the user application (it is not the top layer of the stack) then

its callback function will be handled by the Module above. If there is a Console Interface Module that requires a UART Interface Module then the Console Module will control and use the UART's callback function. In this case the user does not need to create a callback function for the UART Module in their application code.

SSP Interface API Structure

All Interfaces include an API structure which contains function pointers for all the supported Interface functions. An example structure, with the comments removed, for the Digital to Analog Convert (DAC) is shown below.

```
typedef struct st_dac_api
{
    ssp_err_t (* open)(dac_ctrl_t * p_ctrl, dac_cfg_t const * const p_cfg);
    ssp_err_t (* close)(dac_ctrl_t * p_ctrl);
    ssp_err_t (* write)(dac_ctrl_t * p_ctrl, dac_size_t * p_value);
    ssp_err_t (* start)(dac_ctrl_t * p_ctrl);
    ssp_err_t (* stop)(dac_ctrl_t * p_ctrl);
    ssp_err_t (* versionGet)(ssp_version_t * p_version);
} dac_api_t;
```

The API structure is what allows for Modules to easily be swapped in and out for other Modules that are Instances of the same Interface. Let's look at an example application using the DAC Interface above.

Synergy MCUs have an internal DAC peripheral. If the DAC API structure in the DAC Interface were not used, then the application could make calls directly into the module. In the example below the application is making calls to the R_DAC_Write function which is provided in the r_dac module.

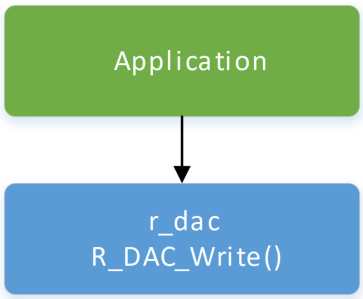


Figure 6: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU by adding a

new external DAC module named r_dac_external. The external DAC uses I2C for communications. The application must now distinguish between the two modules, which adds complexity and further dependencies to the application.

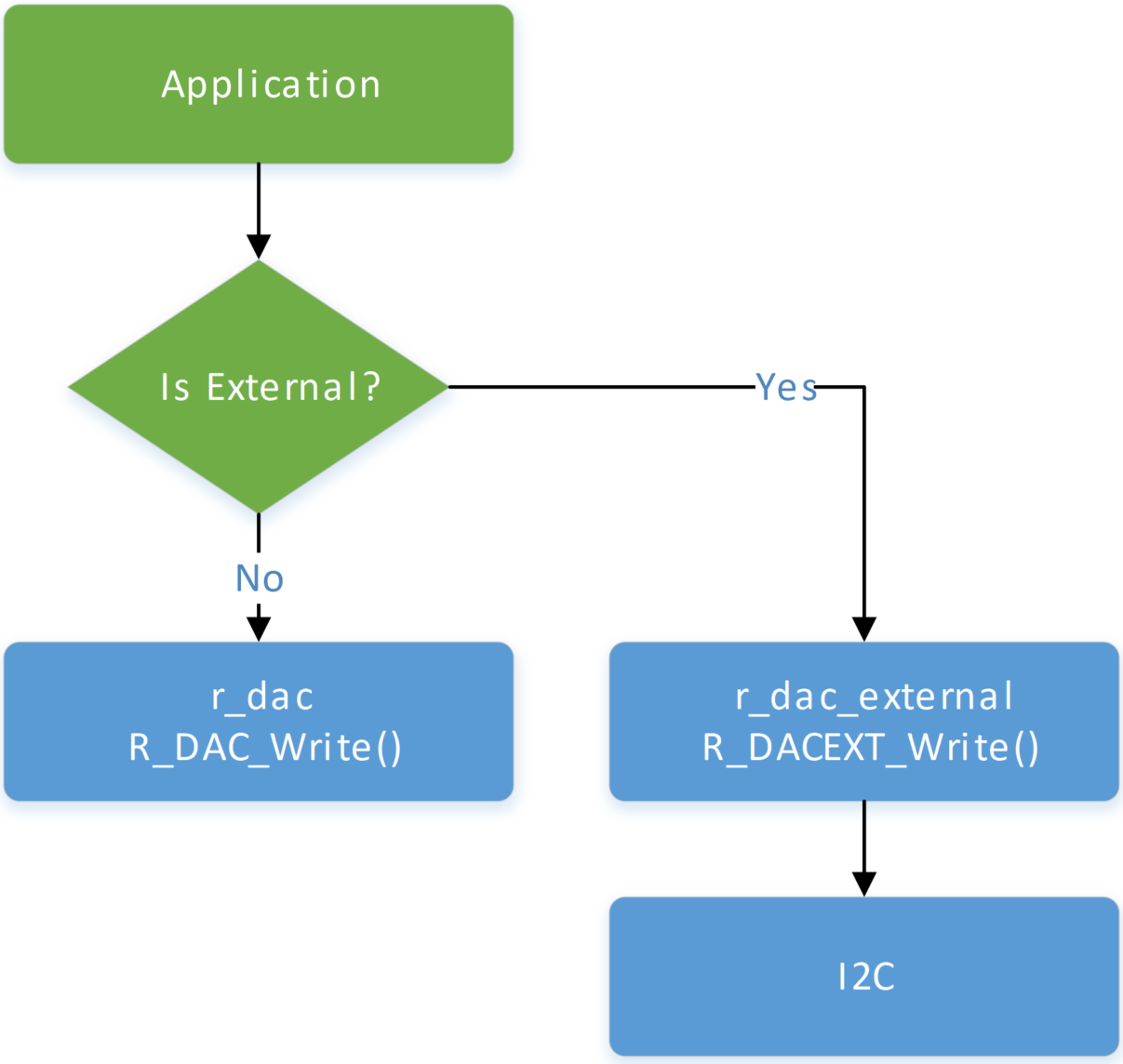


Figure 7: DAC Write with two write modules

The use of Interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed and the application no longer depends upon certain hard-coded Modules. Instead the application now depends on the DAC Interface API which can be implemented by any number of Modules.

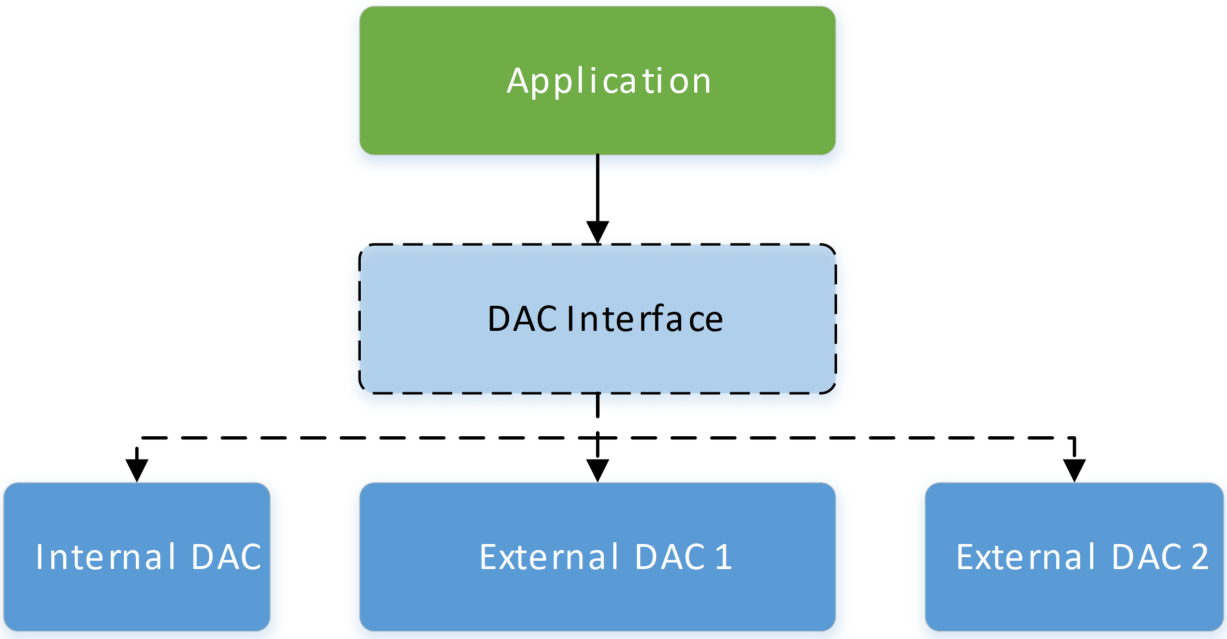


Figure 8: DAC Interface

Functions inside of the API structures follow common names. Most Modules will have a pair of open() and close functions. The open() function must be called before any of the other functions. The only exception is the versionGet() function which is not dependent upon any user provided information.

Other functions that will commonly be found are read(), write(), get(), and set(). Function names are designed to be a noun followed by a verb. Example names include:

- read(), write(), writeRead()
- statusGet()
- calendarAlarmSet(), calendarAlarmGet()
- accessWindowSet(), accessWindowClear()

SSP Interface Version Information

All Interfaces supply a versionGet() function. This function fills in a structure of type ssp_version_t. This structure is made up of two versions: one for the Interface (the API) and one for the underlying Instance that is currently being used.

```
/* Common version structure */
typedef union st_ssp_version
{
    /* Version id */
    uint32_t version_id;
```

```
    /* Code version parameters */
    struct
    {
    uint8_t code_version_major; //  Code major version
    uint8_t code_version_minor; // Code minor version
    uint8_t api_version_major; // API major version
    uint8_t api_version_minor; // API minor version
    };
} ssp_version_t;
```

The API version ideally never changes, and only rarely if it does. A change to the API may require users to go back and modify their code. The code version, the version of the current Instance, may be updated more frequently. Bug fixes, enhancements, and additional features may all bump the code version. Changes to the code version will only require changes to the user code if the user code is using extended features provided by the Instance.

SSP Instances

While Interfaces dictate the features that are provided, Instances actually implement those features. Each Instance is tied to a specific Interface. Instances use the enumerations, data structures, and API prototypes from the Interface. This allows for an application that uses an Interface to swap out the Instance when needed.

On Synergy MCUs some peripherals will have a one-to-one mapping between the Interface and Instance, while others will have a one-to-many. In the example below the IIC and SPI peripherals map to only one Interface each while the SCI peripheral implements three Interfaces.

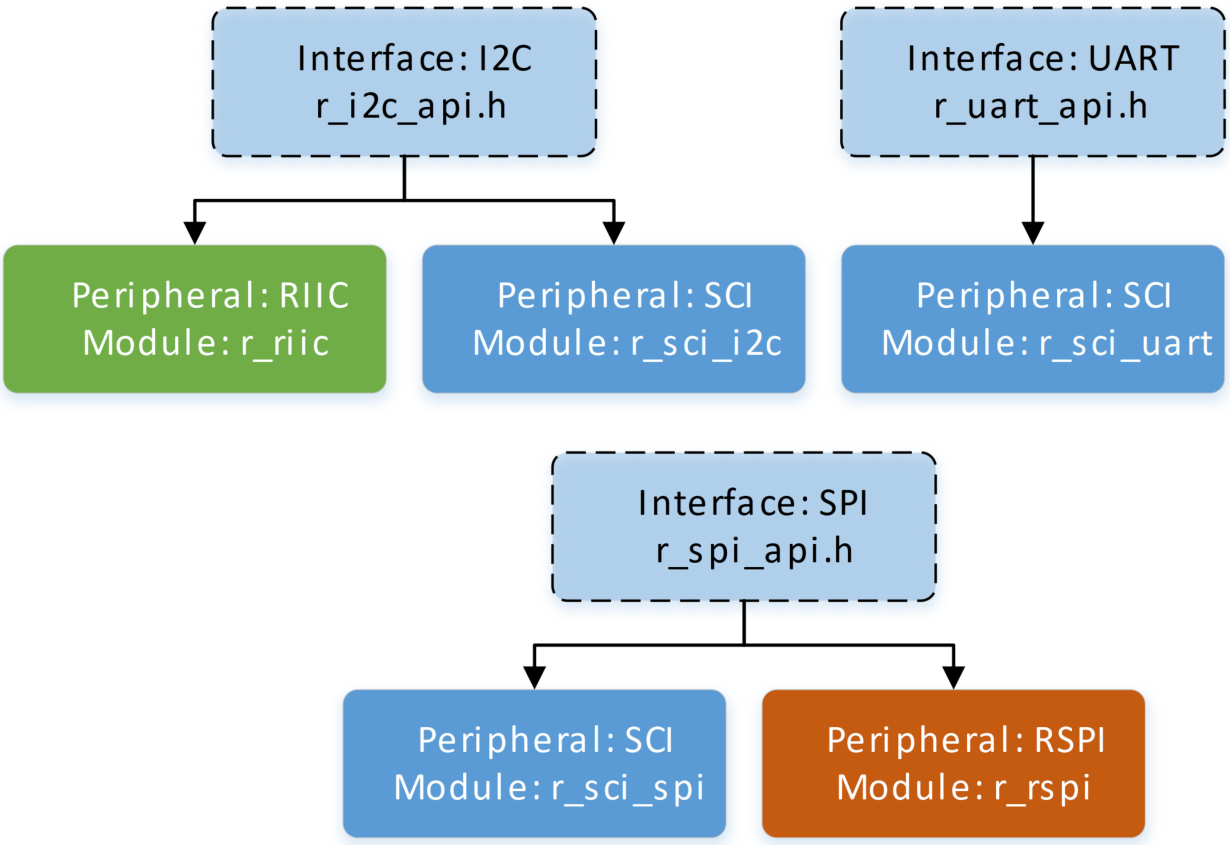


Figure 9: Instances

SSP Instances API Structure

Each Instance includes a constant global structure with its functions that implement the Interface's API. The name of this structure is standardized as g__on_. Examples include g_spi_on_spi, g_transfer_on_dtc, and g_adc_on_adc. This structure is available to be used through an extern in the instances header file (r_spi.h, r_dtc.h, and r_adc.h respectively).

2.1.2.5 Build Time Configuration

All modules have a build-time configuration header file. Most configuration options are supplied at run time. Some options that are rarely used, or apply to all instances of a module, may be moved to build time. The advantage of using a build-time configuration option is to potentially reduce code size reduction by removing an unused feature. Performance enhancements are also possible. All modules have at least one build time option, which is whether to enable or disable parameter checking for the module. SSP modules check function arguments for validity when possible. You may want to disable this feature when your testing has concluded to save code space and to speed up execution.

2.1.2.6 Interface Extensions

In some cases, Instances require more information than is provided in the Interface. This situation can occur in the following two cases:

- An Instance offers extra features that are not common to most Instances of the Interface.
- An Interface must be very generic out of necessity. As an Interface becomes more generic, the number of possible Instances increases. A prime example of this is the Block Media Interface.

```
typedef struct st_sf_block_media_cfg
{
    uint32_t block_size; // Block size in bytes
    void * p_extend; // Instance dependent configuration
} sf_block_media_cfg_t;
```

The configuration structure for the Block Media Interface is intentionally sparse. This allows for nearly endless Instances. Possible Instances include SD card, SPI Flash, SDRAM, USB, and many more. Different configuration information is needed for each Instance. This is accomplished by supplying the information through the p_extend parameter. While the configuration data provided in the p_extend is not the same between Instances, the API calls thereafter will be. This means that the change is only required in one place.

Use of Interface extensions is not always necessary. Some Instances do not offer an extension since all functionality is provided in the Interface. In many cases the p_extend member can be set to NULL. If NULL is provided and the Instance does offer an extension then the Instance will take this to mean that the default options should be used. The documentation for each Instance indicates whether an Interface extension is provided and whether its use is mandatory or optional.

2.1.2.7 SSP Predefined Layers

The SSP comes with two predefined layers: the Driver layer and the Framework layer. The layers are easily identifiable because the modules reside in different folders and have different prefixes. Driver layer modules are located in the ssp/src/driver folder, while Framework level modules are located in the ssp/src/framework folder. Modules in the Driver layer start with an r_ prefix, while Framework level modules start with a sf_ prefix.

The core difference in the functionality between the layers is that Driver layer modules are restricted to being peripheral drivers that are RTOS aware, but do not use any RTOS objects or make any RTOS API calls. This means that Driver layer modules can be used in applications with, or without, an RTOS.

Framework layer modules are free to use RTOS objects such as semaphores, mutexes, or event flags. Framework modules may also create their own when needed. Framework layer modules that need to access hardware typically do so through a Driver layer Interface. Exceptions can be granted in special cases where multiple peripherals need to be used together in a way that would not be practical through multiple individual Interfaces.

2.1.2.8 SSP File Structure

The high-level file structure of the SSP is shown below.

```
ssp
+---inc
\|  +---bsp
\|  \|  +---cmsis
\|  +---driver
\|  \|  +---api
\|  \|  \\---instances
\|  \\---framework
\|  +---api
\|  +---el
\|  +---instances
\|  \\---tes
---src
  +---bsp
  \|  +---cmsis
  \|  \\---mcu
  +---driver
  \|  \\---r_module
  \\---framework
  \\---sf_module
synergy_cfg
+---ssp_cfg
  +---bsp
  +---driver
  \\---framework
}
```

Directly underneath the base ssp folder the folders are split into the source and include folders. Include folders are kept separate from the source for easy browsing and easy setup of include paths. The same set of folders are located in the ssp/inc and ssp/src folders: bsp, driver, and framework.

Apart from the BSP, the SSP's two predefined layers, Driver and Framework, are present. Driver layer modules are located in the ssp/src/driver folder and Framework layer modules are located in ssp/src/framework. Under the include tree, the Driver and Framework layer folders contain two folders each: api and instances. The api folder contains the Interface header files for that layer. The instances folder contains the Instance header files for that layer. Both layers are flat internally which limits the number of include paths required for a project.

The ssp_cfg folder is where configuration header files are stored for each module. Its layout is the same as the ssp folder where the BSP, Driver, and Framework layers have separate flat directories. See the Build Time Configuration section for information on what is provided in these header files.

2.1.2.9 SSP Connecting Layers

SSP modules are meant to be both reusable and stackable. It is important to remember that modules are not dependent upon other modules, but upon other Interfaces. The user is then free to fulfill the Interface using the Instance that best fits their needs.

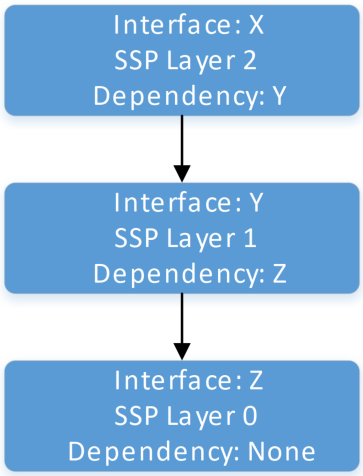


Figure 10: Connecting layers

In the image above Interface Y is a dependency of Interface X and has its own dependency on Interface Z. Interface X only has a dependency on Interface Y. Interface X has no knowledge of Interface Z. This is a requirement for ensuring that layers can easily be swapped out. This is shown in the diagram below:

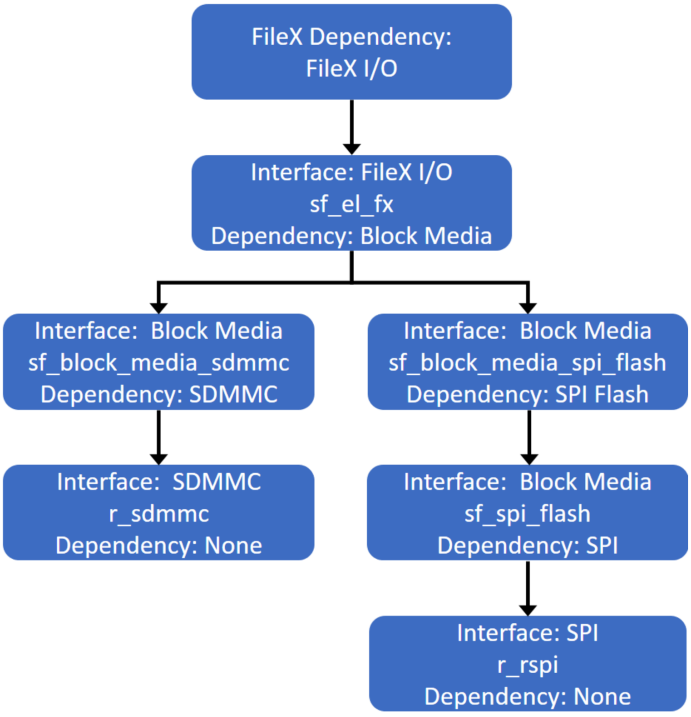


Figure 11: Connecting layers with the FileX interface

In this example we are using the Azure RTOS. FileX file system on two storage mediums: SDMMC and SPI Flash. The SPI Flash Interface takes care of the SPI flash protocol but requires a SPI Interface for actual SPI bus communications. The SDMMC Interface takes care of the protocol and the bus communications meaning that it does not have any dependencies.

2.1.2.10 SSP Architecture In Practice

Each layer in the SSP Stack is responsible for calling the API functions of its dependencies. This can also be described by saying that users are only responsible for calling the API functions at the layer at which they are interfacing. Using the FileX example above, the user is only responsible for calling FileX functions in the application code. Internally, FileX then calls FileX I/O, which in turn calls a Block Media Interface Module. The Block Media Interface can call multiple drivers. At a minimum an upper layer Module calls the open() function of the Interface it depends upon.

To write an application using a Module, you must determine the following:

1) Determine which open() function to call. Dependencies are based upon Interfaces which means that a Module must have some way of discerning which Instance to call.

2) Determine the configuration parameters. The Module also needs to know what configuration information to pass down. In some cases the Module requires certain configuration parameters to be set. If this is the case then the module sets these configuration structure members itself before passing on the rest of the structure. The rest of the configuration structure members must be provided outside of the Module.

3) Provide a control structure which is Module Instance specific and therefore can be allocated by the upper layer module.

Putting this all together means that to interact with a Module Instance, the following is needed:

- A pointer to the Instance's API structure
- A pointer to the Module Instance's configuration structure
- A pointer to the Module Instance's control structure

This information is sufficient to use any module. Notice that the API structure is the only structure that is Instance specific; not Module Instance specific. This is because the API structure will not vary between multiple uses of the same Instance. If SPI is being used on SCI channels 0 and 2 then both Module Instances will use the same API structure while the configuration and control structures will vary.

To make Module Instances easier to use, all of these pieces are encapsulated in instance structures found in each Interface. These structures have a standardized name of <interface>_instance_t. An example from the WDT interface is shown below.

```
typedef struct st_wdt_instance
{
    wdt_ctrl_t * p_ctrl; // Pointer to the control structure for this instance
    wdt_cfg_t const * p_cfg; // Pointer to the configuration structure for this
instance
 wdt_api_t const * p_api; // Pointer to the API structure for this instance
} wdt_instance_t;
```

Upper layer modules that have a dependency on an Interface can then use the instance structure to hold everything needed to interact with an Instance of that Interface. Continuing with the WDT example above, below is the Thread Monitor Framework Interface configuration structure. The Thread Monitor Interface is dependent upon the WDT Interface.

```
typedef struct st_sf_thread_monitor_watchdog_type
{
    wdt_instance_t * p_lower_lvl_wdt; // Pointer to lower level watchdog instance
    bool profiling_mode_enabled; // Enables or disables profiling mode
    UINT priority; // Priority of thread monitor thread
} sf_thread_monitor_cfg_t;
```

The Thread Monitor module has everything it needs to work with the WDT Interface in the *p_lower_lvl_wdt* structure member.

In some cases module dependencies are not be defined in the Interface, but instead in the Instance. An example is the Block Media Interface could be implemented on SDMCC, SPI Flash, or many other Instances (also see API Reference section). Because of the wide range of implementations, the instance structure for a particular Interface cannot be used directly in the Block Media Interface's configuration structure. The Block Media Interface's configuration structure is shown again below.

```
typedef struct st_sf_block_media_cfg
{
    uint32_t block_size; // Block size in bytes
    void * p_extend; // Instance dependent configuration
} sf_block_media_cfg_t;
```

Notice there are Instance structure pointers provided. The reason for this, as previously mentioned, is that the Block Media Interface is too generic to enforce a dependency upon a particular Interface. When a Module is an Instance of a generic Interface, such as Block Media, and it has dependencies on other Modules, the module puts the lower-layer pointers in an extension structure that is referenced through the Interface's *p_extend* configuration member. This is required to allow Module stacking while not forcing Interfaces to expand and have many optional configuration members.

```
typedef struct st_block_media_on_sdmmc_cfg
{
    sdmmc_instance_t const * const p_lower_lvl_sdmmc; // Pointer to SDMMC instance
structure
} sf_block_media_on_sdmmc_cfg_t;
```

2.1.2.11 Using SSP Modules

This section will give general information on how to use a SSP module.

Pick an Interface

Start by picking an Interface for the functionality that is required. For example, for UART communications use the UART Interface.

Find a suitable Instance of the Interface

After picking an Interface, choose a suitable Instance. The list of known Instances of an Interface is

listed in the documentation comments for an Interface. Include the header file of the selected Instance in the source file of the application that uses the Instance.

Allocate control and configuration structures

The e^2 studio ISDE provides a graphical user interface for setting the parameters of the Interface and Instance configuration structures. The ISDE also automatically includes those structures, once they are configured in the GUI, in application-specific header files that you can include into your application code.

To see how the ISDE handles the configuration, see Configuring a Project in the ISDE User's Guide: Using the e2 studio ISDE

The configuration and control structure types follow standard names of <interface>_ctrl_t and <interface>_cfg_t respectively. The ISDE allocates storage for both structures in the application specific header files, which the ISDE creates. Use the ISDE Properties view to set the values for the members of the configuration structure as needed. Many members will be typed enumerations in which case the enumeration can be referenced for available options.

If the Interface has a callback function option, then you first need to declare and define the function in their source code. The return value is always of type void and the parameter to the function is a typed structure of name <interface>_callback_args_t. Once the function has been defined, assign its name to the p_callback member of the configuration structure. If any context information is required in the callback, then the user can provide a pointer to the p_context member. You can assign callback function names through the ISDE Properties window for the selected Module.

Refer to the Instance documentation to see if an Interface extension is provided. If so, then it will be found in the Instance's header file and named <interface>_on__cfg_t. It may have several members just like the Interface's configuration structure. When you select a driver with a specific Instance, you can select any parameter in the configuration structure of the instance in the ISDE property.

Interact using Interface's Instance Structure

Once the instance structure has been defined, you can interact with the Instance as needed. Below is code that builds up an instance structure for the UART Interface as implemented on SCI. Please note that when using e^2 studio for Synergy, the following code is automatically generated for the user.

```
/* Include the header file of the Instance. */
#include "r_sci_uart.h" // This will in turn include the r_uart_api.h Interface
/* Allocate control structure. */
uart_ctrl_t my_uart_ctrl;
/* Setup extended UART configuration on SCI. */
uart_on_sci_cfg_t my_uart_extended_cfg =
{
    /* Set extended configuration members... */
};
/* Configure standard UART Interface. */
uart_cfg_t my_uart_cfg =
```

```
{
    .data_bits = UART_DATA_BITS_8,
    /* Continue configuring other members... */
    .p_extend = &my_uart_extended_cfg
};
/* Setup instance structure */
uart_instance_t my_uart = {
    .p_ctrl = &my_uart_ctrl,
    .p_cfg = &my_uart_cfg, //Extended configuration is brought through
in p_extend
    .p_api = &g_uart_on_sci //Defined in r_sci_uart.h
};
```

Now that the instance structure is ready, you can interact with the UART Interface. In e^2 studio, the name of the instance structure is the *Name* that you provide when configuring the Module Instance in the ISDE Properties window.

```
ssp_err_t err;
/* Initialize UART */
err = my_uart.p_api-&gt;open(my_uart.p_ctrl, my_uart.p_cfg);
/* Check return for errors. */
if (SSP_SUCCESS != err)
{
/* Handle error. */
}
/* Use other Interface functions. */
err = my_uart.p_api-&gt;write(my_uart.p_ctrl, ...);
err = my_uart.p_api-&gt;read(my_uart.p_ctrl, ...);
```

2.1.2.12 Coding Style

C99 Use

SSP uses the ISO/IEC 9899:1999 (C99) C programming language standard. Specific features introduced in C99 that are used include standard integer types (stdint.h), booleans (stdbool.h), designated initializers, and the ability to intermingle declarations and code.

Use of const in API parameters

The const qualifier is used with API parameters whenever possible. An example case is shown below:

```
ssp_err_t (* open)(flash_ctrl_t * const p_ctrl, flash_cfg_t const * const p_cfg);
```

While not fool-proof by any means, this does provide some extra checking inside the SSP code to ensure that arguments that should not be altered are treated as such.

Weak Symbols

Weak symbols are used occasionally in and with SSP. They are used to ensure that a project builds even when you have not defined an optional function.

2.1.3 BSP Architecture

This section describes the BSP or Board Support Package. For the API Reference see Board Support Package. The BSP is board specific and as a result also MCU specific.

2.1.3.1 What Does the BSP Do?

The BSP is responsible for getting the MCU from reset to the user's application (that is, the main() function). Before reaching the user's application the BSP sets up the stacks, heap, clocks, interrupts, and C runtime environment. The BSP also configures and sets up the port I/O pins and performs any board specific initializations.

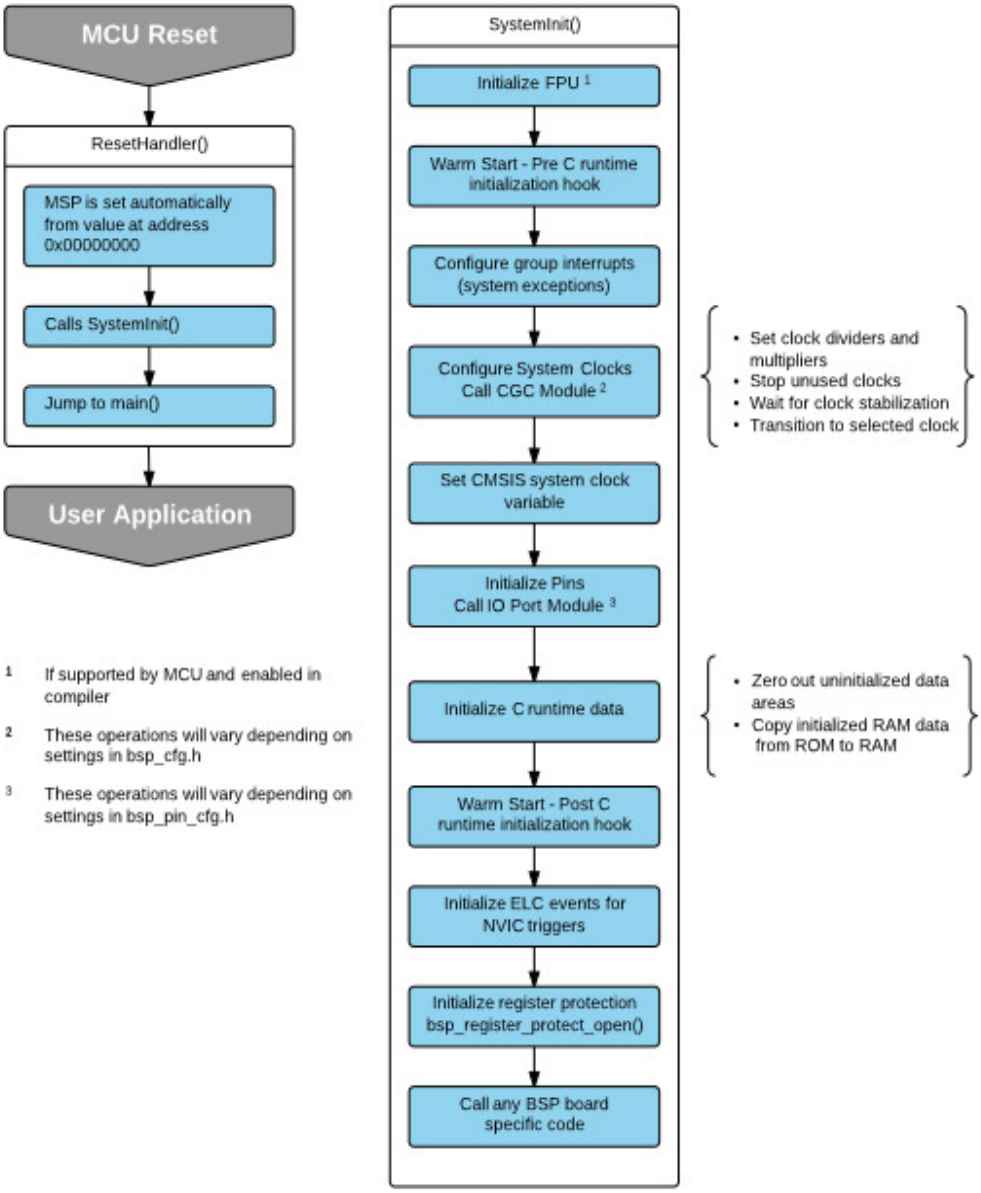


Figure 12: BSP flow

2.1.3.2 BSP Related Terminology

Term	Meaning
system_xxxx.c or startup_xxxx.c	The 'xxxx' refers to the MCU type. For example, system_S7G2.c when referencing the S7G2 MCU.
BSP	Short for Board Support Package. BSP's usually have source files related to a specific board.

Callback Function	This term refers to a function that is called when an event occurs. For example, the NMI interrupt handler is implemented in the BSP. The user will likely want to know when an NMI system exception occurs. To alert the user, a callback function can be configured for the group interrupts (a group of exceptions all of which are tied to the NMI). When an NMI occurs the BSP will jump to the provided callback function and the user can handle the error. Interrupt callback functions should be kept short and be handled carefully because when they are called the MCU will still be inside of an interrupt and therefore will be delaying any pending interrupts.

2.1.3.3 BSP Directory Structure

The BSP is organized into folders containing MCU, board specific and CMSIS information.

Synergy is CMSIS-compliant and based on the CMSIS-Core. This requires that we follow CMSIS requirements and naming standards.

- Standardized definitions for processor peripherals
- NVIC (Nested Vector Interrupt Controller)
- Systick (System Tick Timer)
- MPU (Memory Protection Unit)
- Standardized access functions to access processor features
- NVIC_SetPriority()
- NVIC_EnableIRQ
- Standardized function names for system exception handlers
- Reset_Handler()
- SysTick_Handler()
- Standardized functions for system initialization.
- SystemInit() – defined in system_S7G2.c for S7G2 MCUs
- Standardized software variables for clock speed information
- SystemCoreClock

The BSP directory structure is shown below:

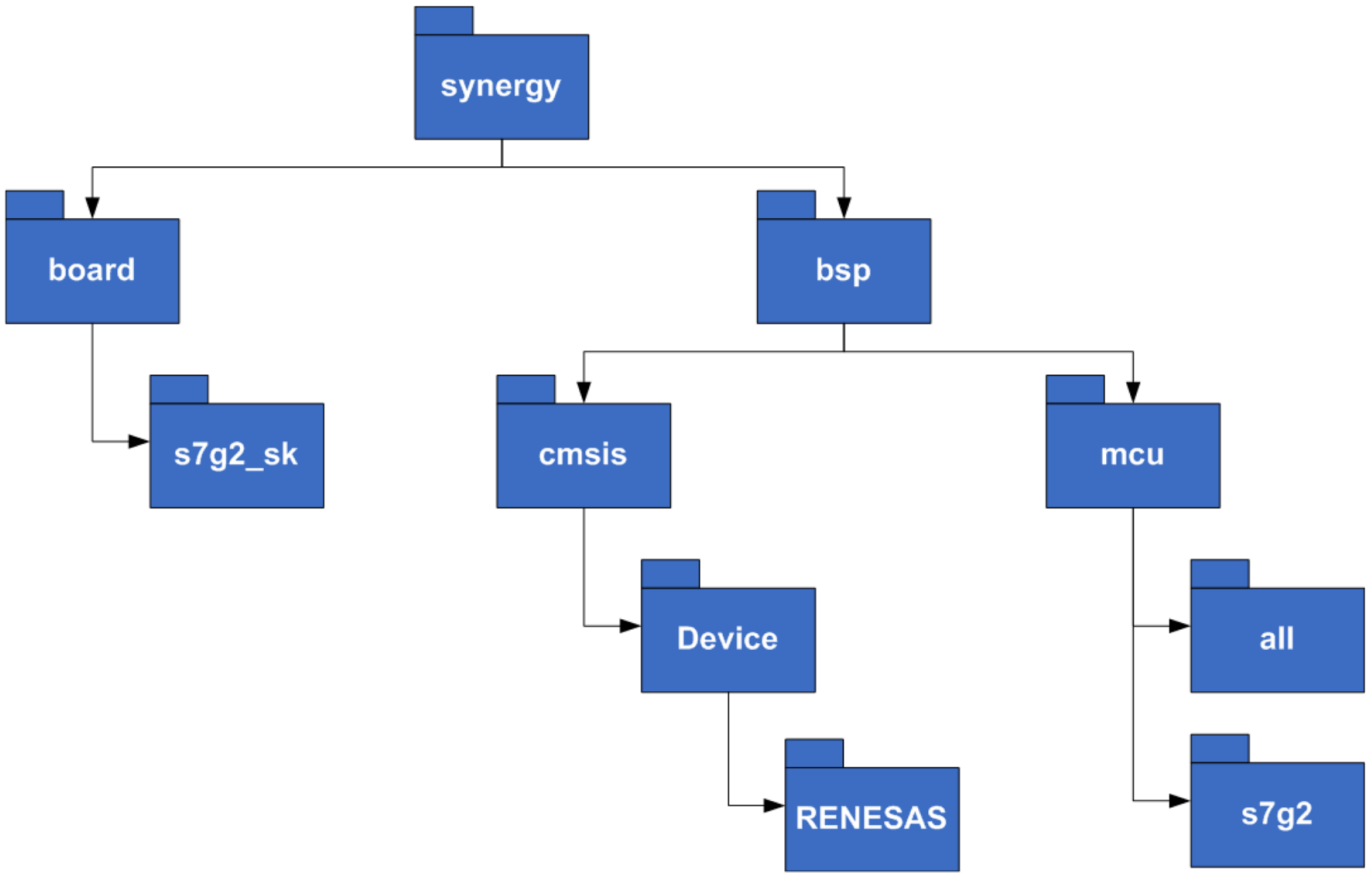


Figure 13: BSP directory structure

2.1.3.4 Configuring the BSP

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated by the ISDE when the Generate Project Content button is clicked.

2.1.3.5 BSP Configuration Settings

The table below describes each of the configurable BSP settings. Many of these settings are MCU specific and there are differences between the settings available for each of the supported MCUs.

Table: BSP Configuration options

BSP Property	Description
Part number	MCU part number
ram_size_bytes	Available RAM in this MCU package
rom_size_bytes	Available ROM in this MCU package
data_flash_size_bytes	Available Data Flash in this MCU package
package_style	Style of package (ie. BGA)
package_pins	Number of pins in this MCU package

series	MCU part series
Main stack size (bytes)	Size of the Main Stack. Must be > 0.
Process stack size (bytes)	Size of the Process Stack. Use of this stack is optional. If 0, then PSP use is disabled
Heap size (bytes)	Size of the heap in bytes. If 0, the heap is disabled.
OFS0 register settings: IWDT Start Mode, IWDT Timeout Period, IWDT Dedicated Clock Frequency Divisor, IWDT Window End Position, IWDT Window Start Position, IWDT Reset Interrupt Request Select, IWDT Stop Control, WDT Start Mode Select , WDT Timeout Period, WDT Clock Frequency Division Ratio, WDT Window End Position, WDT Window Start Position, WDT Reset Interrupt Request, WDT Stop Control	The option-setting memory determines the state of the MCU after a reset. It is allocated to the configuration setting area and the program flash area of the flash memory. See the MCU user manual for details.
OFS1 register settings: Voltage Detection 0 Circuit Start, Voltage Detection 0 Level, HOCO Oscillation Enable. S3 MPU has MPU configuration settings.	See the MCU user manual for details.
MPU - Enable or disable PC Region 0	Start block address for access window protection
MPU - PC0 Start, MPU - PC0 End, MPU - Enable or disable PC Region 1, MPU - PC1 Start, MPU - PC1 End, MPU - Enable or disable Memory Region 0, MPU -Memory Region 0 Start, MPU - Memory Region 0 End, MPU - Enable or disable Memory Region 1, MPU - Memory Region 1 Start, MPU - Memory Region 1 End, MPU - Enable or disable Memory Region 2, MPU - Memory Region 2 Start, MPU - Memory Region 2 End, MPU - Enable or disable Memory Region 3, MPU - Memory Region 3 Start, MPU - Memory Region 3 End	Secure MPU ROM register settings. See user manual for details.
ID code 1, ID code 2, ID code 3, ID code 4	Sets the ID Code for boot mode and debugger access protection.
MCU Vcc (mV)	Some Modules (e.g. LVD) need to know the voltage supplied to the MCU. This information is obtained from here.
Parameter checking	Defines whether the global setting for parameter checking is enabled or disabled. Local modules will take this value by default but can be locally overridden.
Assert Failures	Defines what happens when an assertion failure occurs.
Error Log	Defines whether or not errors are logged to ssp_error_log.

2.1.3.6 BSP Configuration Files

Configuration files are used by the BSP to set up ROM registers, clocks, interrupts, ELC events and initial pin configurations. These configuration files can be found in ssp_cfg\bsp.

Bsp_cfg.h

This configuration file contains the values for BSP system settings. These are the settings that can be modified from the ISDE BSP properties tab. They include ROM register settings, stack and heap size, parameter checking and control of error logging.

Some registers are located in ROM and therefore must be set at compile-time. These include some option-setting memory (OFS) registers as well as certain memory protection registers.

Option-setting memory determines the state of the MCU after a reset. For example, the IWDT can be configured and enabled, voltage detection can be enabled, and HOCO oscillation can be enabled. When these registers are set the operations are completed before the MCU's reset vector is fetched and execution begins.

Some Synergy MCUs include a Memory Protection Unit (MPU). The MPU is a programmable device that can be used to define memory access permissions (i.e. privileged access only or full access) and memory attributes (for example, bufferable, cacheable) for different memory regions. The MPU can support up to eight programmable memory regions, each with their own programmable starting addresses, sizes and settings.

The ISDE configures these memory areas by setting values for the provided MPU settings. You must be careful when setting these registers. Incorrect settings can prevent access to required memory areas or prevent access to the MCU entirely.

2.1.3.7 BSP Pin Configuration

You can configure the pins used in your application through the ISDE pin configurator. See Configuring Pins.

Bsp_pin_cfg.h

This configuration file contains an array of pin configurations. During start-up, and before main() is executed, the BSP iterates over this array and initializes the MCU's port pins based on the settings in the array. Initially, before any pin configuration by the user, the ISDE **Pins** tab displays the initial reference configuration defined for the selected board type (see Configuring Pins). Once the user modifies the pin configuration and clicks **Generate Project**, a new bsp_pin_cfg.h file is generated containing the new pin configuration. The BSP always uses the bsp_pin_cfg.h file from ssp_cfg\bsp as the source for its pin configuration information, but the pin information generated by clicking **Generate Project** is written to a bsp_pin_cfg.h file in the hidden folder ssp_cfg\bsp\.out.

In this way, the user can manually edit the bsp_pin_cfg.h in ssp_cfg\bsp without the fear of the file being overwritten by the project generation, while the Pin Configuration information generated by the ISDE also remains available for view or merging with the user's config file.

2.1.3.8 BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in bsp_clock_cfg.h. These settings are derived from clock configuration information provided from the ISDE **Clocks** tab setting.

- Clock configuration is performed prior to initializing the C runtime environment to speed up

the startup process, as it is possible to start up on a relatively slow (for example, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.

Bsp_clock_cfg.h

This configuration file represents the values for system clock settings. These are the settings that can be modified from the ISDE **Clocks** tab. See: Configuring Clocks

2.1.3.9 System Interrupts

As Synergy MCU's are based on the Cortex-M ARM architecture, the NVIC Nested Vectored Interrupt Controller (NVIC) handles exceptions and interrupt configuration, prioritization and interrupt masking. In the ARM architecture, the NVIC handles exceptions. Some exceptions are known as System Exceptions. System exceptions are statically located at the top of the vector table and occupy vector numbers 1 to 15. Vector zero is reserved for the MSP Main Stack Pointer (MSP). The remaining 15 system exceptions are shown below:

- Reset
- NMI
- Cortex-M4 Hard Fault Handler
- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCall Handler
- Cortex-M4 Debug Monitor Handler
- Reserved
- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI and Hard Fault exceptions are enabled out of reset and have fixed priorities. Other exceptions have configurable priorities and some can be disabled.

2.1.3.10 Group Interrupts

Group interrupt is the term used to describe the 12 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the Synergy peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDT Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- VBATT monitor Interrupt
- Oscillation Stop is detected

- NMI pin
- RAM Parity Error
- RAM ECC Error
- MPU Bus Slave Error
- MPU Bus Master Error
- MPU Stack Error

A user may enable notification for one or more group interrupts by registering a callback using the BSP API function R_BSP_GroupIrqWrite. When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

As mentioned earlier, the first 16 slots in the vector table are already accounted for by the system exceptions. Beginning with slot 16 are user configurable interrupts. These may be external, or peripheral generated interrupts.

The size of the NVIC interrupt table varies across Synergy MCU types (shown below).

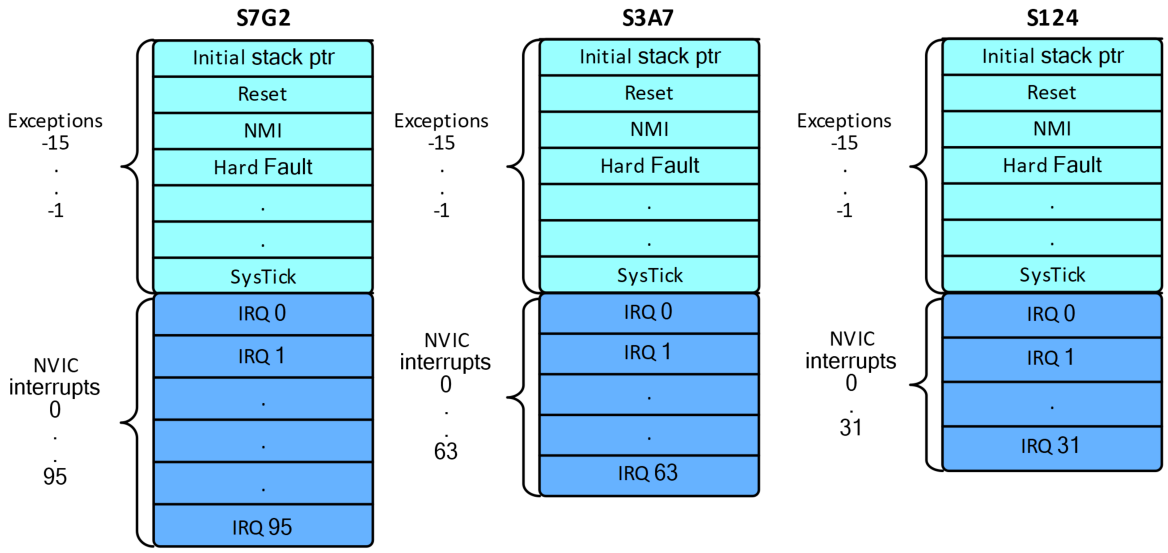


Figure 14: NVIC Interrupt vector table

Although the number of available slots for the NVIC interrupt vector table may seem small, the BSP defines up to 512 events that are capable of generating an interrupt. By using Event Mapping, the BSP maps user enabled events to NVIC interrupts. For an S7G2 MCU, only 96 of these events may be active at any one time, but the user has flexibility by choosing which events generate the active event.

The diagram below shows the interrupt vector table for the S7G2:

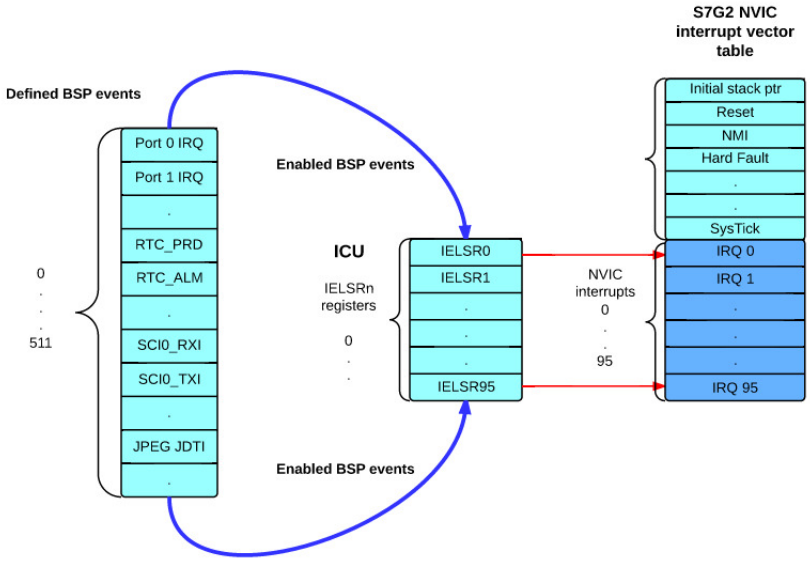


Figure 15: NVIC Interrupt vector table

By allowing the user to select only the events they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and event specific.

For example, on other microcontrollers a standard NVIC interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the Synergy implementation there is a vector entry for each of the SCI0 events that we are interested in. The difference between a standard NVIC table and the Synergy S7G2 NVIC table is shown below:

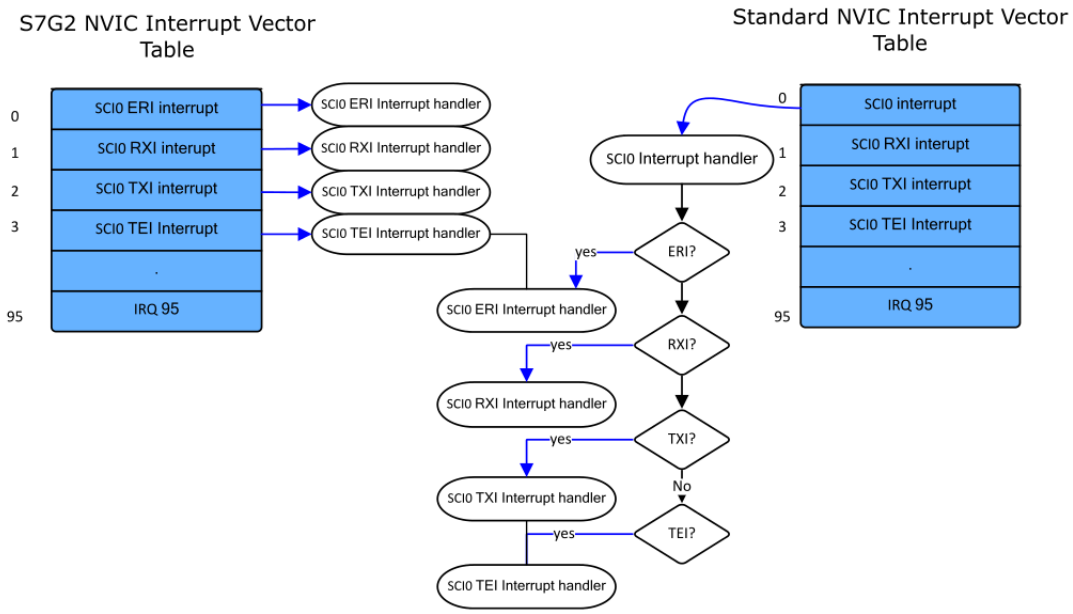


Figure 16: NVIC Interrupt vector table example

Configuration of interrupts is handled by the ISDE. Selecting an interrupt to be used by a module will generate the code necessary to allocate its entry in the vector table, as well as link it to the proper ICU ELC event.

When an interrupt occurs one of the very first operations must be to call R_BSP_IrqStatusClear() with the interrupt number corresponding to the NVIC interrupt slot that was assigned by the BSP. R_BSP_IrqStatusClear() clears the interrupt status flag (IR) for the given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.

Entries that have been assigned a priority (i.e. BSP_IRQ_CFG_ICU_IRQ0) in our example, have their corresponding 'weak handler' address placed in the next available vector slot. All of the possible interrupt sources are iterated over in this manner. Defined interrupts are entered into the vector table.

BSP Interrupt Configuration File

Bsp_irq_cfg.h is a legacy file that is no longer used and will be removed in the future. Interrupt configuration is now completely handled by the ISDE.

Vector table entries

System exceptions such as the HardFault_Handler, are defined as weak references. This allows the user to override the default handler for a particular exception and define their own handler.

All other entries in the vector table are generated by the ISDE using a macro which defines the vector and a corresponding vector information structure to generate an entries in ROM table linker sections (.vector.* for vectors and .vector_info.* for vector information).

Note that in CMSIS system_xxxx.c, there is also a weak definition (and a function body) for the Warm Start callback function R_BSP_WarmStart(). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by you by copying the body into your application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

Warm start callbacks

As the BSP is in the process of bringing up the board out of reset, there are two points where the user can request a callback. These are defined as the 'Pre C' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as R_BSP_WarmStart(), so it is a simple matter of redefining the function or copying the existing body from CMSIS system_xxxx.c into the application code to get a callback. R_BSP_Warmstart() takes an event parameter which describes the type of warm start callback being made.

```
/* Different warm start entry locations in the BSP. */
typedef enum e_bsp_warm_start_event
{
    BSP_WARM_START_PRE_C = 0, // Called almost immediately after reset.
```

```
    /* No C runtime environment, clocks, or IRQs. */
    BSP_WARM_START_POST_C // Called after clocks and C runtime environment have been
set up.
} bsp_warm_start_event_t;
```

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system_xxxx. To use this function just copy this function into your own code and modify it to meet your needs.

Pre C Warm start callback

This callback occurs almost immediately after reset and at this point no C runtime environment, clocks, or IRQs have been setup.

Why would you be interested in a 'Pre C' warm start callback?

Below are a few examples.

- Execution of safety code (i.e. destructive memory tests) as part of the startup process.
- Examination of global memory as part of a crash dump investigation.
- Preventing re-initialization of an already running RTC.

Post C Warm start callback

This callback occurs after clocks and the C runtime environment have been setup.

Why would you be interested in a 'Post C' warm start callback?

Below are a few examples.

- Run tests that require that clocks have been setup.
- ADC diagnostics.
- ROM/External memory system checks.

2.1.3.11 Custom BSP Board support

Creating a Custom BSP for your own board is not supported in this version of the SSP. For information on creating a Custom BSP with earlier versions, see application note R11AN0071EU, *Creating a Custom Board Support Package for SSP v1.2.0 or Later*.

2.1.3.12 BSP API functions

The BSP provides public functions, available to any project using the BSP, that allow access to functionality that is common across BSP supported MCUs and boards.

- R_BSP_SoftwareLockInit: The BSP provides API functions to implement atomic locking. These locks can be used to protect critical areas of code as an RTOS semaphore or mutex normally would. This function simply initializes a defined lock structure to BSP_LOCK_UNLOCKED
- R_BSP_SoftwareLock: Attempts to acquire the lock that has been sent in. The Load-

Exclusive and Store-Exclusive instructions are being used to perform an exclusive read-modify-write on the input lock. This process is:
    - Use a load-exclusive (LDREXB) to read the value of the lock.
    - If the lock is available, then modify the lock value so it is reserved. If not available, then issue CLREX.
    - Use a store-exclusive to attempt to write the new value back to memory.
    - Test the returned status bit to see if the write was performed or not.
- R_BSP_SoftwareUnlock: Releases the hold on an existing software lock.
- R_BSP_HardwareLock: Hardware locks are similar to Software locks. In fact, the BSP Software lock functions are called by the Hardware lock functions. Hardware locks are specific to a particular peripheral, the list of available hardware locks being defined in bsp_hw_locks.h. Hardware locks can be used to prevent multiple threads from trying to use a peripheral that is already in use by a process or thread. For example, when the Flash API open() function is called, it takes the Flash Hardware lock and keeps it until the Flash API close is called.
- R_BSP_HardwareUnlock: Releases the hold on an existing hardware lock. In the Flash example above, the Flash API close function would call this function.
- R_BSP_GroupIrqWrite: Registers a callback function for one of supported group interrupts. As described earlier, there are 12 of these and they are all mapped to the NMI exception. When an NMI occurs, the NMI_Handler looks at the NMISR (status register) to determine the source of the interrupt. If a callback function has been registered for this group interrupt, it will be called. If NULL is passed for the callback argument, then any previously registered callbacks are unregistered.
- R_BSP_IrqStatusClear: Clears the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR, then the interrupt will trigger again immediately.
- R_BSP_SoftwareDelay: Implements a blocking software delay. A delay can be specified in microseconds, milliseconds, or seconds. The delay is implemented based on the system clock rate.
- R_BSP_VersionGet: Returns the version of the BSP.
- R_BSP_LedsGet(): Returns information about the LEDs on the board.
- R_BSP_ModuleStop(): Specifies modules whose stop bit should be set.
- R_BSP_ModuleStart(): Specifies modules whose stop bit should be cleared.
- R_BSP_CacheOff(): Turns off the ROM cache, and return it's prior state.
- R_BSP_CacheSet(): Sets the cache state to a specific state (on or off).
- R_BSP_RegisterProtectEnable: Enables register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). The registers that may be protected are grouped together into one of three groups.
    - BSP_REG_PROTECT_CGC - registers related to the clock generation circuit.
    - BSP_REG_PROTECT_OM_LPC_BATT - registers related to operating modes, low power consumption, and battery backup function.
    - BSP_REG_PROTECT_LVD – registers related to LVD (Low Voltage Detection)

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function does not have its register protection settings inadvertently modified.

- Each time RegisterProtectDisable() is called, the respective reference counter is incremented.
- Each time RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

As the example below shows, without reference counters, MODULE2 would re-protect the registers that MODULE 1 had un-protected, preventing MODULE1 from writing them.

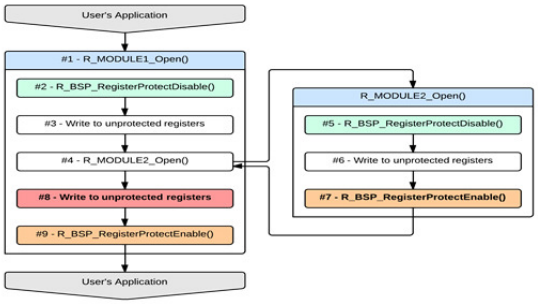


Figure 17: Register protection

- R_BSP_RegisterProtectDisable: Disables register protection. Registers that are not protected can be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). The register groupings described above still apply.

2.1.4 Key Features

This section describes the key features

2.1.4.1 Azure RTOS ThreadX® RTOS

- Multithreaded, deeply embedded, real-time systems
- Small, fast Picokernel™ architecture
- Multitasking capabilities
- Preemptive and cooperative scheduling
- Flexible thread priority support (32-1024 priority levels)
- Small memory footprint and fast response times
- Optimized interrupt handling
- Stack Pointer Overflow Monitor

2.1.4.2 Azure RTOS GUIX™

- Supports 2D Graphics Acceleration in Hardware
- Unlimited objects (screens, windows, widgets)
- Dynamic object creation/deletion
- Alpha blending and anti-aliasing at higher color depths
- Canvas blending
- Dithering support
- Complete windowing support, including viewports and Z-order maintenance
- Multiple canvases and physical displays
- Window blending and fading
- Screen transitions, sprites, and dynamic animations
- Touchscreen and virtual keyboards
- Multilingual support with UTF8 string encoding
- Automatic size scaling

- 8-bit Color Lookup Table (CLUT) support
- Touch Rotation
- Radial Progress Bar
- Endian Neutral
- Monochrome through 32-bit true-color with alpha graphics formats
- Skinning and Themes
- Bitmap compression
- GUIX Studio desktop tool and Win32 simulation
- Integrated with hardware JPEG/MJPEG decoder

2.1.4.3 Azure RTOS USBX™

- USB 2.0 Full Speed and High-Speed support
- Device class: MSC, HID, CDC-ACM
- Host class: MSC, HID, CDC-ACM, UVC, HUB, Printer, Video
- Supports fast DMA and isochronous transfers

2.1.4.4 Azure RTOS FileX®

- MS-DOS compatible file system integrated with ThreadX
- FAT12-, 16-, 32-bit support
- exFAT
- Fault-tolerant file system (uses journaling)
- Multiple media instances
- LevelX Flash block media driver
- LevelX support for NOR Flash on QSPI

2.1.4.5 Azure RTOS NetX™

- Integrated with wired (Ethernet) and wireless (WiFi, Cellular) networking interfaces for Synergy
- IPv4 compliant TCP/IP Protocol Stack
- Integrated with ThreadX
- Zero-copy API
- UDP Fast Path Technology
- BSD-compatible socket layer
- RFC 791 Internet Protocol (IP)
- RFC 826 Address Resolution Protocol (ARP)
- RFC 903 Reverse Address Resolution Protocol (RARP)
- RFC 792 Internet Control Message Protocol (ICMP)
- RFC 3376 Internet Group Management Protocol (IGMP)
- RFC 768 User Datagram Protocol (UDP)
- RFC 793 Transmission Control Protocol (TCP)
- RFC 1112 Host Extensions for IP Multicasting

2.1.4.6 Application Frameworks

- ADC Periodic framework
- Audio Playback framework
- Audio Playback HW DAC framework
- Audio Playback HW I2S framework
- Audio Record framework
- Audio Recording HW ADC framework
- Block Media Interface for SD Multi Media Card
- Block Media LevelX NOR framework

- Block Media QSPI framework
- Block Media RAM framework
- Block Media SDMMC framework
- Bluetooth Low Energy (BLE) framework
- Deprecated – Capacitive Touch Sensing Unit framework
- Capacitive Touch Sensing Unit framework Version 2
- Deprecated – Capacitive Touch Sensing Unit Button framework
- Deprecated – Capacitive Touch Sensing Unit Slider framework
- Cellular framework
- Communications framework on NetX
- Communications framework on NetX Telnet
- Deprecated - Communications framework on USBX
- Communications Framework on USBX version2
- Console framework
- External Interrupt framework
- I2C framework
- Inter-Thread Messaging framework
- JPEG Decode framework
- Memory framework
- Port LevelX framework
- Periodic Sampling ADC framework
- Power Profile Version 2 framework
- SPI Framework
- Synergy FileX® Port Block Media Interface framework
- Synergy GUIX™ Interface framework
- Synergy NetX™ Port framework
- Synergy USBX™ Port framework
- Thread Monitor framework
- Deprecated - Touch Panel framework
- Touch Panel Version 2 framework
- UART framework
- Wi-Fi Framework

2.1.4.7 Security Cryptographic (SCE) Library

- True RNG (TRNG)
- SHA1, SHA224/SHA256
- ECC P-192, P-224, P-256 and P-384 curves. Includes APIs for scalar multiplication, key generation, ECDSA signature generation, and ECDSA signature verification operations
- AES 128, 192, and 256-bits ECB, CBC, CTR, GCM, XTS
- 3DES, 192-bit key, ECB, CBC, CTR
- ARC4
- RSA up to 2048-bit keys
- DLP, DSA up to 2048-bit keys
- Encryption/Decryption
- Key Generation (plaintext and wrapped keys) and Installation.
- Signature Generation and Verification
- MD5

2.1.4.8 CMSIS DSP Library

- Basic math functions
- Fast math functions
- Complex math functions
- Filters

- Convolution
- Matrix functions
- Transforms
- Motor control functions
- Statistical functions
- Support functions
- Interpolation functions
- Bayes functions
- Controller functions
- Distance functions
- Quaternion functions
- SVM functions

Detailed CMSIS library details can be found on
Github: https://github.com/ARM-software/CMSIS_5/releases/tag/5.8.0

2.1.4.9 CMSIS Neural Network Library

- Convolution functions
- Activation functions
- Fully-connected Layer functions
- Pooling functions
- Softmax functions
- Basic math functions
- Concatenation functions
- NN Support functions
- Reshape Functions
- SVD Functions

Detailed CMSIS library details can be found on
Github: https://github.com/ARM-software/CMSIS_5/releases/tag/5.8.0

2.1.4.10 AzureRTOS NetX Duo™

- IPv4 and IPv6 compliant TCP/IP Protocol Stack
- Integrated with ThreadX
- Integrated with wired (Ethernet) and wireless (WiFi, Cellular) networking interfaces for Synergy
- Zero-copy API
- UDP Fast Path Technology
- BSD-compatible socket layer
- RFC 2460 IPv6 Specification
- RFC 4861 Neighbor Discovery for IPv6
- RFC 4862 IPv6 Stateless Address
- RFC 1981 Path MTU Discovery for IPv6
- RFC 4443 ICMPv6
- RFC 791 Internet Protocol (IP)
- RFC 826 Address Resolution Protocol (ARP)
- RFC 903 Reverse Address Resolution Protocol (RARP)
- RFC 792 Internet Control Message Protocol (ICMP)
- RFC 3376 Internet Group Management Protocol (IGMP)
- RFC 768 User Datagram Protocol (UDP)
- RFC 793 Transmission Control Protocol (TCP)
- RFC 1112 Host Extensions for IP Multicasting
- RFC 1661 - The Point-to-Point Protocol (PPP)

2.1.4.11 Azure RTOS NetX™ Applications (IPv4 Networking Services)

- DHCP Client and Server
- DNS Client
- HTTP 1.0 Client and Webserver
- HTTP 1.1 Client
- FTP Client and Server
- TFTP Client and Server
- Telnet Client and Server
- Auto IP
- NAT
- SMTP Client
- POP3 Client and Server
- SNMP Agent
- SNTP Client
- PPP (Not currently supported by Synergy Configuration tool)

2.1.4.12 Azure RTOS NetX Duo™ Applications (IPv4/v6 Networking Services)

- DHCP Client and Server
- DNS Client
- HTTP 1.0 Client and Webserver
- HTTP 1.1 Client
- HTTPS Client and Server
- FTP Client and Server
- TFTP Client and Server
- Telnet Client and Server
- Auto IP
- NAT
- SMTP Client
- POP3 Client and Server
- SNMP Agent
- SNTP Client
- MDNS/DNS-SD
- PPP (Not currently supported by Synergy Configuration tool)

2.1.4.13 Azure RTOS NetX Secure

- TLS v1.2 (RFC 5246) and v1.3 (RFC 8446)
- DTLS v1.2 (RFC 6347)
- RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 5280 X.509 PKI Certificates (v3)
- Supports X.509 extensions for Key Usage and Extended Key Usage
- RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Transport Layer Security (TLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 8422 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier
- RFC 4279 Pre-Shared Key Cipher suites for TLS
- Supports TLS extensions for:
    - Secure Renegotiation Indication

    - Server Name Indication
    - Signature Algorithms
- Subject Alternative Name
- Integrated with hardware accelerated Cryptographic library on Synergy

2.1.4.14 Azure RTOS MQTT client for NetX Duo

- Compliant with OASIS MQTT Version 3.1.1
- Provides option to enable/disable TLS for secure communications using NetX Secure in SSP
- Supports QoS and provides the ability to choose the levels that can be selected while publishing the message
- Supports multiple Instances of MQTT Client Per Device

2.1.4.15 Memory Support

- Flash programming support via JTAG
- Code and Data Flash drivers
- External memory bus support

2.1.4.16 Human Machine Interface (HMI)

- Graphics LCD controller driver
- Segment LCD controller driver
- Capacitive Touch Sensing Unit (CTSU)

2.1.4.17 Hardware Abstract Layer (HAL) Driver Modules

- Analog Comparator High-Speed (ACMPHS)
- Analog Comparator Low Power (ACMPLP)
- Analog Connect Module (ACM)
- Analog to Digital Converter (ADC) (12-bit, 14-bit)
- Asynchronous General Purpose Timer (AGT)
- AGT Input Capture (AGT Input Capture)
- Capacitive Touch Sensing Unit (CTSU)
- Clock Frequency Accuracy Measurement (CAC)
- Clock Generation Circuit (CGC)
- Controller Area Network Interface (CAN)
- Cyclic Redundancy Check calculator (CRC)
- Data Operation Circuit (DOC)
- Data Transfer Controller (DTC)
- Digital to Analog converter (DAC)
- Digital to Analog converter 8-bit (DAC8)
- Direct Memory Access Controller (DMAC)
- Event Link Controller (ELC)
- Flash Memory-High Performance (FLASH_HP)
- Flash Memory-Low Power (FLASH_LP)
- General Purpose I/O Port (GPIO / IOPORT)
- General Purpose Timer (GPT)
- General PWM Timer with Input Capture (GPT_INPUT_CAPTURE)
- Graphics LCD Controller (GLCD)
- IEEE 1588 Precision Time Protocol (PTP)
- I2IC (RIIC)
- Independent Watchdog Timer (IWDT)
- Interrupt Controller Unit (ICU)
- JPEG Codec (JPEG_COMMON, JPEG_ENCODE, JPEG_DECODE)

- Keyboard Interrupt Interface (KINT)
- Deprecated – Low Power Mode (LPM)
- Low Power Mode Version 2 (LPMv2)
- Low Voltage Detection (LVD)
- Parallel Data Capture Unit (PDC)
- Quad SPI (QSPI)
- Real Time clock (RTC)
- SD Multi Media Card (SDMMC)
- Segment LCD (SLCD)
- Serial Communication Interface I2C (SCI_I2C)
- Serial Communication Interface SPI (SCI_SPI)
- Serial Communication Interface UART (SCI_UART)
- Sigma-Delta ADC (SDADC)
- Serial Peripheral Interface (SPI)
- Serial Sound Interface (SSI)
- Watchdog Timer (WDT)

2.1.4.18 GPIO and Key Interrupts

- GPIO module
- Key Interrupts module

Chapter 3 Starting Development

To start development with the Renesas Synergy Software Package (SSP), download and install e2 studio, obtain a target Synergy development or evaluation board, and run through the tutorials in this chapter. The e2 studio ISDE user guide and the tutorials include step-by-step instructions for getting started with a simple application. To get started with the SSP, refer to these pages:

- e2 studio ISDE User Guide
- Tutorial: Your First Synergy Project - Blinky
- Tutorial: Using HAL Drivers - Programming the WDT
- IAR Embedded Workbench for Renesas
- What is Synergy Standalone Configurator (SSC)?

3.1 e2 studio ISDE User Guide

3.1.1 Using the e2 studio ISDE

This section describes how to use the Renesas e^2 studio Integrated Solutions Development Environment (ISDE) to develop applications with the Renesas Synergy Software Package (SSP). The architecture of the SSP directly determines how you use the e^2 studio ISDE to develop a Synergy application. See the following documents for details on the SSP architecture included in this manual:

- SSP Architecture
- BSP Architecture

For simple example projects generated and built with e^2 studio, see:

- Tutorial: Using HAL Drivers - Programming the WDT
- Tutorial: Your First Synergy Project - Blinky

All User Guides in this manual show how to configure a driver and develop an application using the e^2 studio ISDE. See:

- HAL Layer for HAL layer user guides
- Framework Layer for Framework layer user guides

3.1.2 What is the e2 studio ISDE?

The Renesas e^2 studio ISDE, or Integrated Solution Development Environment, is a development tool encompassing code development, build, and debug. The ISDE is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT). Specifically for Synergy MCUs, the ISDE provides a Graphical User Interface (GUI) with numerous wizards for configuring and auto-generating

code using the Synergy Software Package (SSP). The ISDE also incorporates a smart manual so that driver and device documentation is available in the form of tooltips right in the code.

Figure 18: e2 studio Splash Screen

Note

The e² studio screens shown in this manual are examples. Some details may differ between different releases of the e² studio ISDE and the SSP.

The e² studio ISDE and the Synergy Project Configurator have been developed to make it as easy as possible to quickly select the SSP modules required for a particular application, include them in a project, and configure them. The ISDE provides a graphical user interface to configure all elements of the SSP for the Synergy MCU applications. In addition to HAL and Framework modules, the ISDE can add and configure RTOS threads, semaphores, mutexes, event flags, and queues. This makes adding RTOS support to an application very straightforward. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required. The ISDE generates the complete and correct configuration code from the selections in the configuration views, so you can focus on writing the application code.

The elements of the SSP are shown in the **Project Explorer** view of the e² studio ISDE. All SSP configuration structures and parameters are mapped to XML files. The XML files enable the ISDE to present a visual list of configurable options that you can select from. In addition to generating code to configure the modules, the XML also provides dependency information for modules.

When you add an SSP module to your project, the e² studio ISDE checks the dependencies of this module and adds all necessary drivers and framework modules to create the appropriate stack. If there is a dependency that requires you to make a choice, this module is highlighted in the Stack window and the ISDE guides your selection by showing the available options.

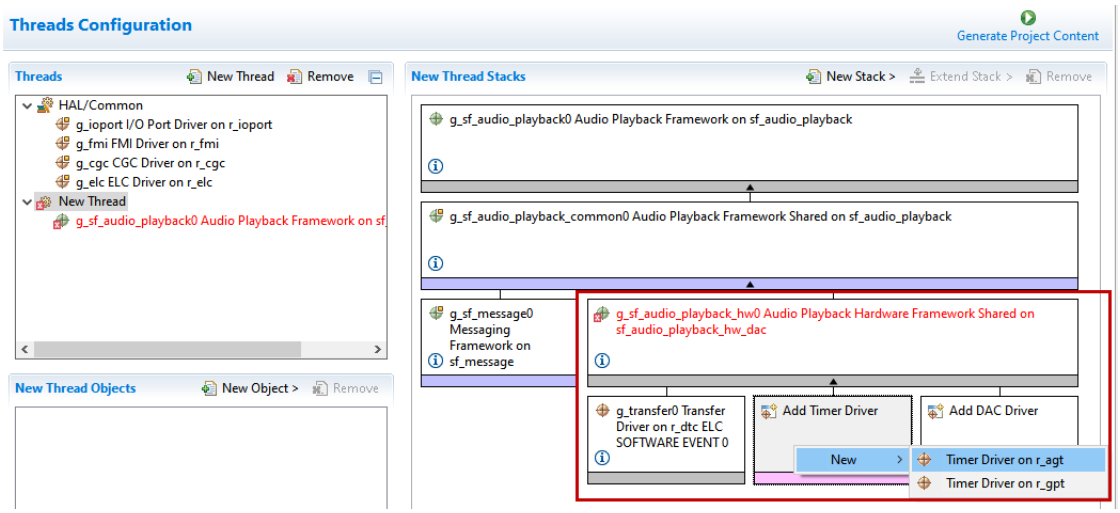


Figure 19: ISDE Dependency Checking

Errors are flagged next to the Driver name in the HAL/Common Modules or New Thread Modules pane. You can also review errors in the Problems window.

3.1.3 e2 studio ISDE Prerequisites

3.1.3.1 Obtaining a Synergy Kit

To develop applications with the SSP, start with one of the Renesas Synergy Kits. The Renesas Synergy Kits are designed to seamlessly integrate with the e^2 studio ISDE. Several types of kits are available:

- Development Kit (DK)
- Starter Kit (SK)
- Product Example (PE)
- Target Board (TB)

Ordering information, Quick Start Guides, User Manuals, and other related documents for all Synergy Kits are available at http://renesassynergy.com.

3.1.3.2 PC Requirements

To use the e^2 studio ISDE, ensure that your PC meets the following minimum requirements:

- Windows 7 with Intel i5 or i7, or AMD A10-7850K or FX
- Memory: 8 GB DDR3 or DDR4 DRAM (16 GB DDR4/2400 MHz RAM is preferred)
- Minimum 250 GB hard disk

3.1.3.3 Installing e2 studio and the SSP

Detailed installation instructions and installers for the e^2 studio ISDE and the SSP are available on the Renesas Synergy Gallery website https://synergygallery.renesas.com. Review the release notes for e^2 studio to ensure that the e^2 studio version supports the selected SSP version.

3.1.3.4 Choosing a Toolchain

The e^2 studio ISDE can work with several toolchains and toolchain versions such as the GNU ARM

compiler and the IAR toolchain. A version of the GNU ARM compiler is included in the e^2 studio installer and has been verified to run with the SSP version.

To use the IAR toolchain for ARM, install IAR Embedded Workbench for Renesas Synergy (EWSYN) (a license from IAR is required). Before starting a Synergy project with IAR, also install the IAR Embedded Workbench for ARM Eclipse plugin (using the IAR Embedded Workbench plugin manager in the 'Help' menu).

3.1.3.5 Adding the IAR Embedded Workbench for Renesas Synergy Compiler into e2 studio

The IAR Embedded Workbench for Renesas Synergy compiler (IAR compiler) can now be used from within ^{2} studio. This allows the developer to have the advantages provided by the IAR compiler without the need to also use the IAR EW for Renesas Synergy ISDE. The installation process involves installing IAR EW for Renesas Synergy, installing e^2 studio, and installing the associated IAR plugins for e^2 studio. The process is described in the application note found with this search: https://www.renesas.com/eu/en/document/apn/installing-iar-compiler-e2-studio-application-note

The application note also includes a description of how to migrate a project that originally used e^2 studio and GCC. It also includes a description of how to migrate a project from IAR 7.x to IAR 8.x so it will be successfully opened when using e^2 studio.

3.1.4 What is a Project?

In e^2 studio, all SSP applications are organized in Synergy projects. Setting up a Synergy project involves:

1. Creating the project
2. Configuring the project

The e^2 studio ISDE has many project wizards and configuration windows specifically for Synergy projects. You can create a new Synergy Project with the **Synergy Project Generator** or edit the configuration of an existing project in the **Synergy Project Editor**.

When you launch e^2 studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** view. Each project has an associated configuration file named configuration.xml which is located in the project's root directory.

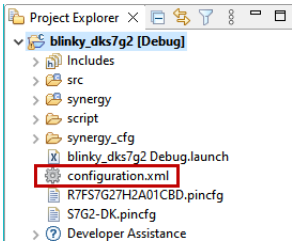


Figure 20: e2 studio Project Configuration File

Double-click on the configuration.xml file to open the Synergy Project Editor and view or modify all configuration settings associated with this project. To edit the project configuration, make sure that the **Synergy Configuration** perspective is selected in the upper right hand corner of the e^2 studio window.

Figure 21: e2 studio Synergy Configuration Perspective

Note

Whenever the Synergy project configuration (that is, the configuration.xml file) is saved, a verbose Synergy Project Report file (synergy_cfg.txt) with all the project settings is generated. The format is such that differences can be easily viewed using a textual difference tool. The generated file is located in the project root directory.

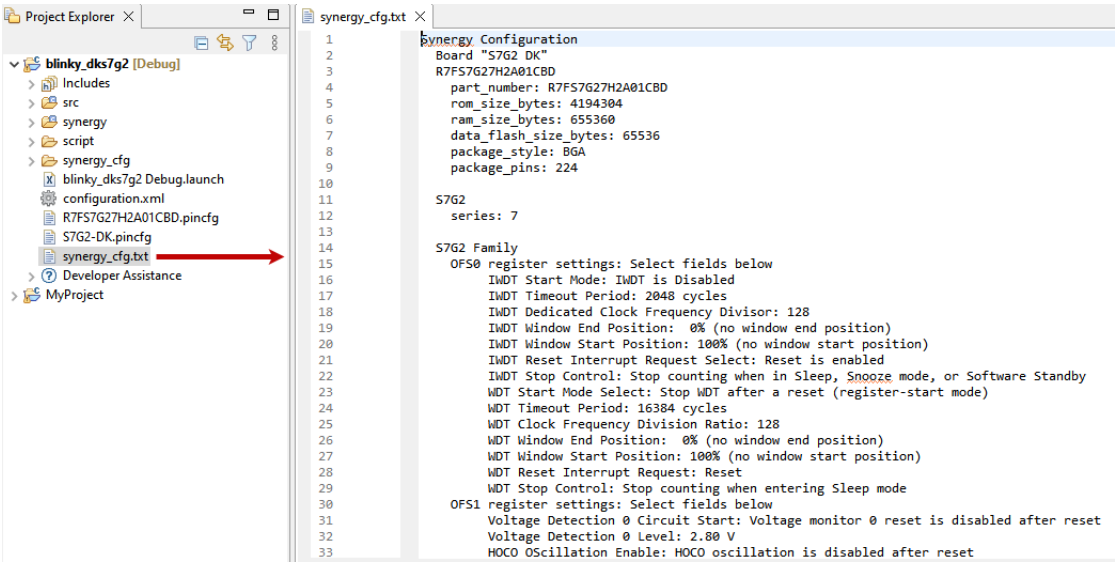


Figure 22: Synergy Project Report

The Synergy Project Editor has a number of tabs. The configuration steps and options for individual tabs are discussed in the following sections.

Note

Which tabs are available with the Synergy Project Editor depends on the e² studio version.

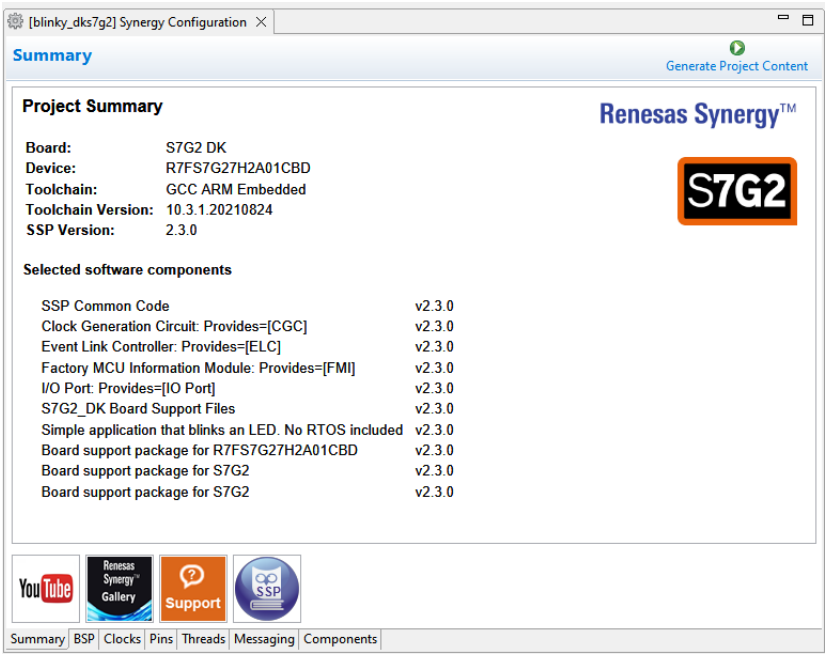


Figure 23: e2 studio Project Editor

3.1.5 Creating a Project

This section includes step-by-step instructions for creating a Synergy Project. Once you have created the project, you can easily configure the hardware (clocks, pins, interrupts) and the parameters of all modules that are part of your application.

To create a new Synergy Project with the Synergy Project Generator, select the project name, select the hardware for your application, select the toolchain and choose from preconfigured clock, pin, and MCU related settings by selecting a project template.

3.1.5.1 Creating a New Project

For Synergy applications, always generate a new project as a Synergy Project in the following way:

1. Click on **File > New > Synergy C/C++ Project**

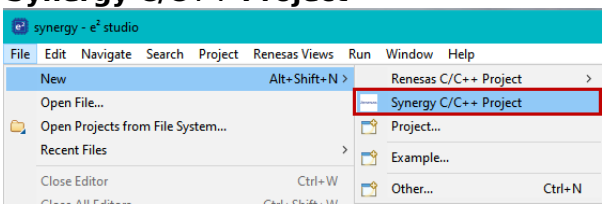


Figure 24: New Synergy Project

Then click on the type of template for the type of project you are creating.

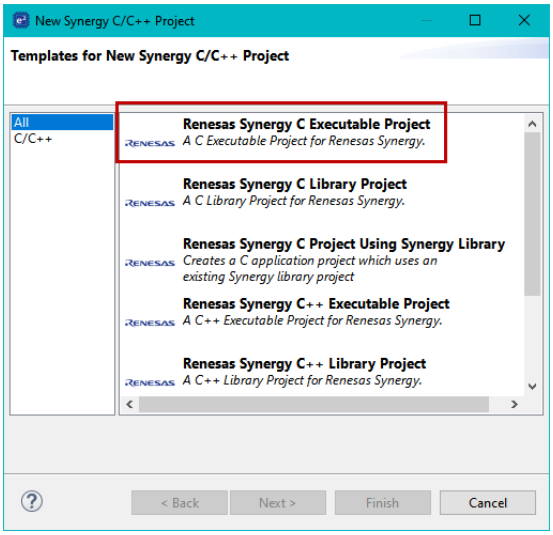


Figure 25: New Project Templates

2. Select a project name and location.

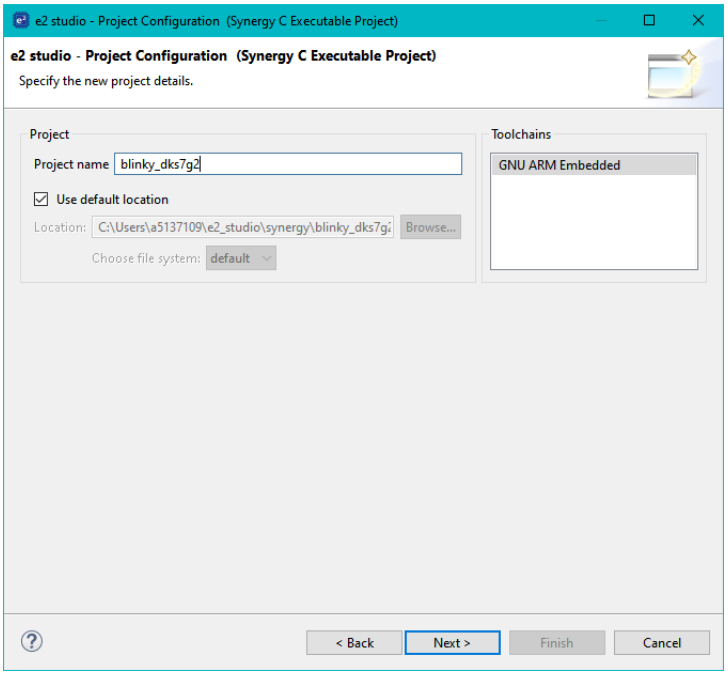


Figure 26: Synergy Project Generator (Screen 1)

3. Click **Next**.

3.1.5.2 Selecting a Board and Toolchain

In the next Project Configuration window select the hardware and software environment:

1. Select the **SSP version**.

2. Select the **Board** for your application. You can select an existing Synergy Kit or select Custom User Board for any of the Synergy devices with your own BSP definition.

Note

To develop your own BSP, see the following Application Note: "Creating a Custom Board Support Package" at http://renesassynergy.com.

3. Select the **Toolchain version**.
4. Select the **Debugger**. The J-Link ARM Debugger is preselected.
5. Click **Next**.

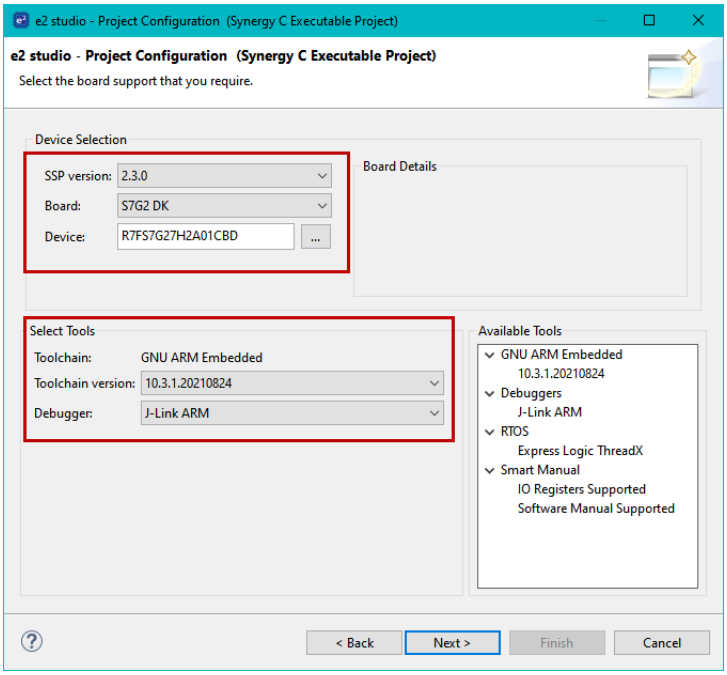


Figure 27: Synergy Project Generator (Screen 2)

3.1.5.3 Selecting a Project Template

In the next window, select a project template from the list of available templates and click **Finish**.

Note

If you want to develop your own application, select a basic template for your board, such as S7G2-DK BSP. You can add RTOS support at any time while you configure the modules for your project.

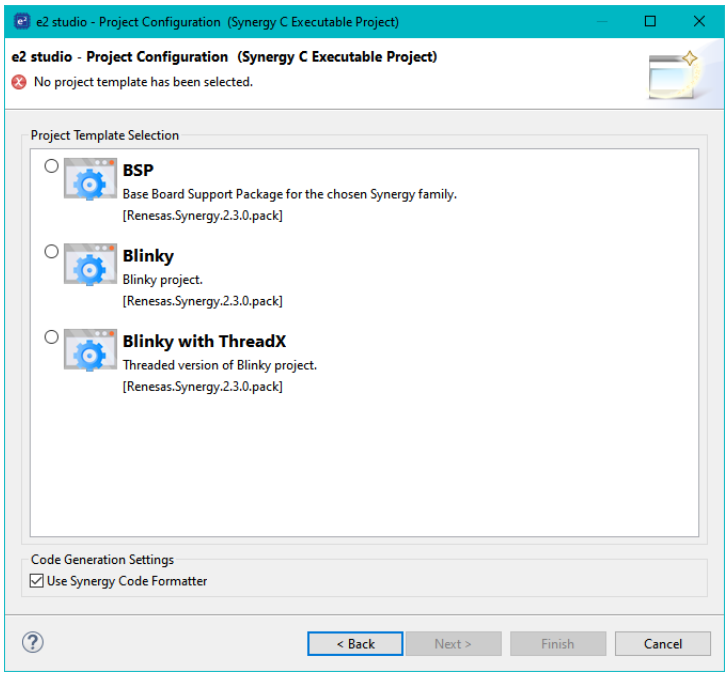


Figure 28: Synergy Project Generator (Screen 3)

By default, this screen shows the templates that are included in your current SSP pack.

When the project is created, the ISDE displays a summary of the current project configuration in the Synergy Project Editor.

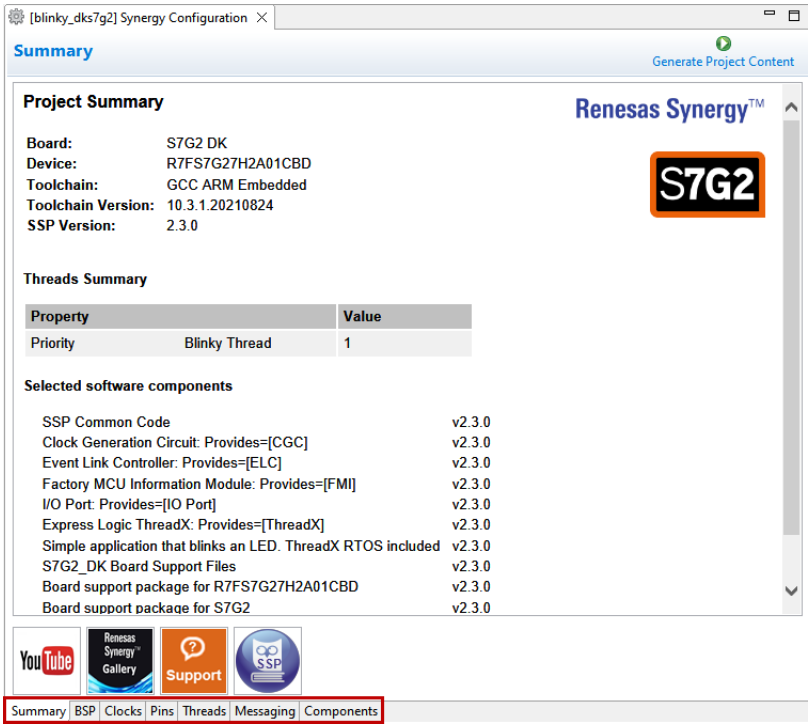


Figure 29: Synergy Project Editor and Available Editor Tabs

On the bottom of the Synergy Project Editor view, you can find the tabs for configuring multiple aspects of your project:

- With the BSP tab, you can change board specific parameters from the initial project selection.
- With the Clocks tab, you can configure the MCU clock settings for your project.
- With the Pins tab, you can configure the electrical characteristics and functions of each port pin.
- With the Threads tab, you can add SSP modules and drivers for RTOS and non-RTOS applications and configure the drivers. For each module or driver selected in this tab, the Properties window provides access to the configuration parameters, interrupt priorities, and pin selections.
- With the Messaging tab, you can configure the Messaging Framework for ThreadX-based projects. The Messaging tab is included in e^2 studio version 5.0 and higher.
- The Components tab provides an overview of the selected modules. You can also add drivers for specific SSP releases and application sample code here.

3.1.6 Configuring a Project

A project has two levels of configuration.

- The BSP, Clocks, and Pins tabs determine the initial configuration of the MCU after reset and before any user code is executed. By selecting a project template during project creation, the ISDE configures default values as appropriate for the selected board. You can change those default values as needed.
- The Threads allows you to add SSP modules to the project and set the configuration parameters of the module as needed by the application. Because the Messaging Framework is an integral part of many ThreadX-based applications, you can configure the Messaging Framework for each thread requiring messaging in the Messaging tab (for e^2 studio versions 5.0 and higher).

3.1.6.1 Configuring the BSP with the ISDE

The **BSP** tab shows the currently selected board (if any) and device. The **Properties** view is located in the lower left of the Project Configurations view as shown below.

Note

*If the Properties view is not visible, click **Window** > **Show View** > **Properties** in the top menu bar.*

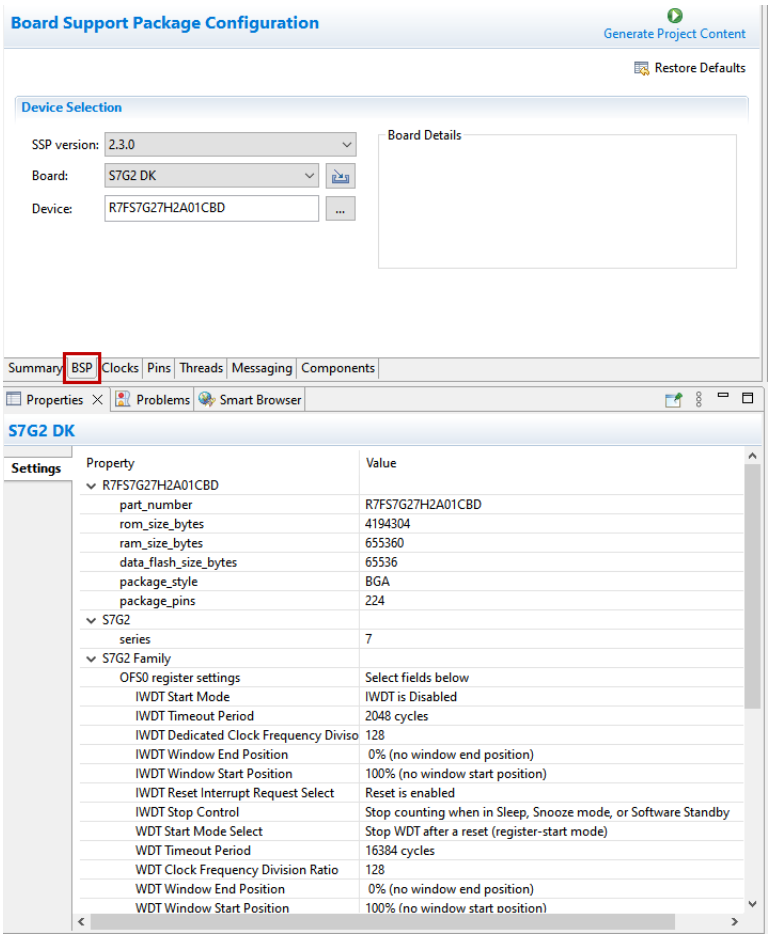


Figure 30: ISDE BSP Tab

The Properties view shows the configurable options available for the BSP. These can be changed as required. The BSP is the SSP layer above the MCU hardware. The ISDE checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you press the **Generate Project Content button**, the BSP configuration contents are written to

synergy_cfg/ssp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

3.1.6.2 Configuring Clocks

The **Clocks** tab presents a graphical view of the MCU's clock tree, allowing the various clock dividers and sources to be modified. If a clock setting is invalid, the offending clock value is highlighted in red. It is still possible to generate code with this setting, but correct operation cannot be guaranteed. In the figure below, the USB clock UCLK divider has been changed so the resulting clock frequency is 60 MHz instead of the required 48 MHz. This parameter is colored red.

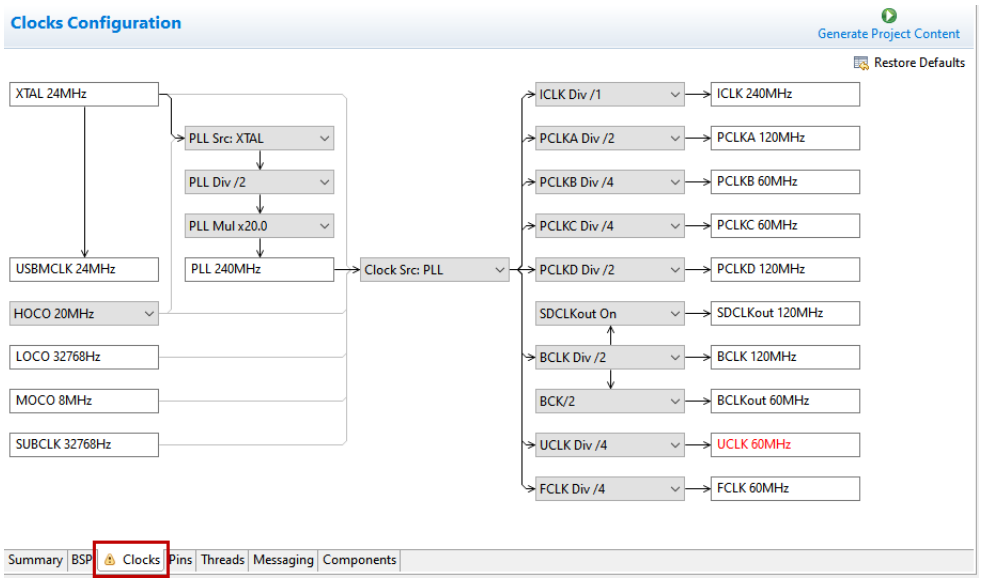


Figure 31: ISDE Clocks Tab

When you press the **Generate Project Content** button, the clock configuration contents are written to:

synergy_cfg/ssp_cfg/bsp/bsp_clock_cfg.h

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

3.1.6.3 Configuring Pins

The **Pins** tab provides flexible configuration of the MCU's pins. As many pins are able to provide multiple functions, they can be configured on a peripheral basis. For example, selecting a serial channel via the SCI peripheral offers multiple options for the location of the receive and transmit pins for that module and channel. Once a pin is configured, it is shown as green in the **Package** view.

Note

*If the Package view window is not open in the ISDE, select **Window** > **Show View** > **Pin Configurator** > **Package** from the top menu bar to open it.*

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as the DK-S7G2, some peripherals connected on the board are preselected.

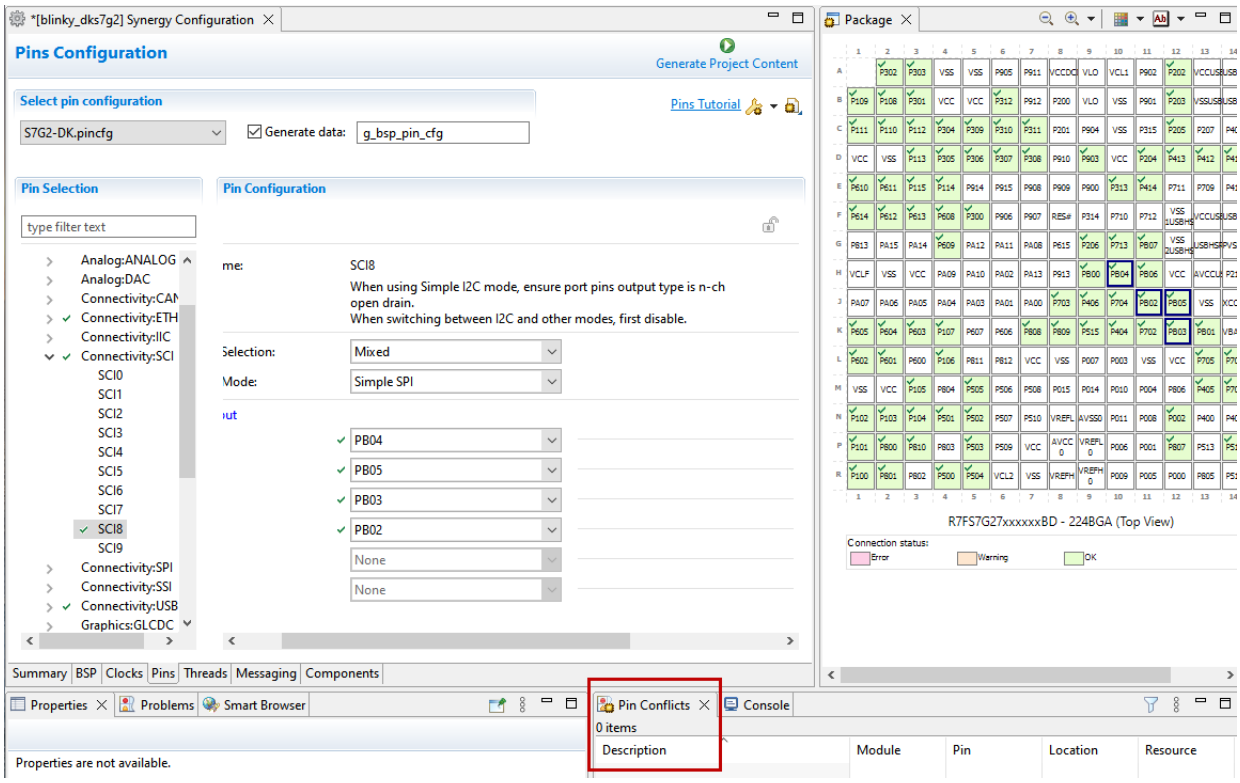


Figure 32: ISDE Pins Tab

The pin configurator includes a built-in conflict checker, so if the same pin is allocated to another peripheral or I/O function the pin will be shown as red in the package view and also with white cross in a red square in the **Pin Selection** pane and **Pin Configuration** pane in the main **Pins** tab. The **Pin Conflicts** view provides a list of conflicts, so conflicts can be quickly identified and fixed.

In the example shown below, port P105 is already used by the External Memory peripheral, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the External Memory peripheral in the **Pin Selection** pane on the left side of the tab.

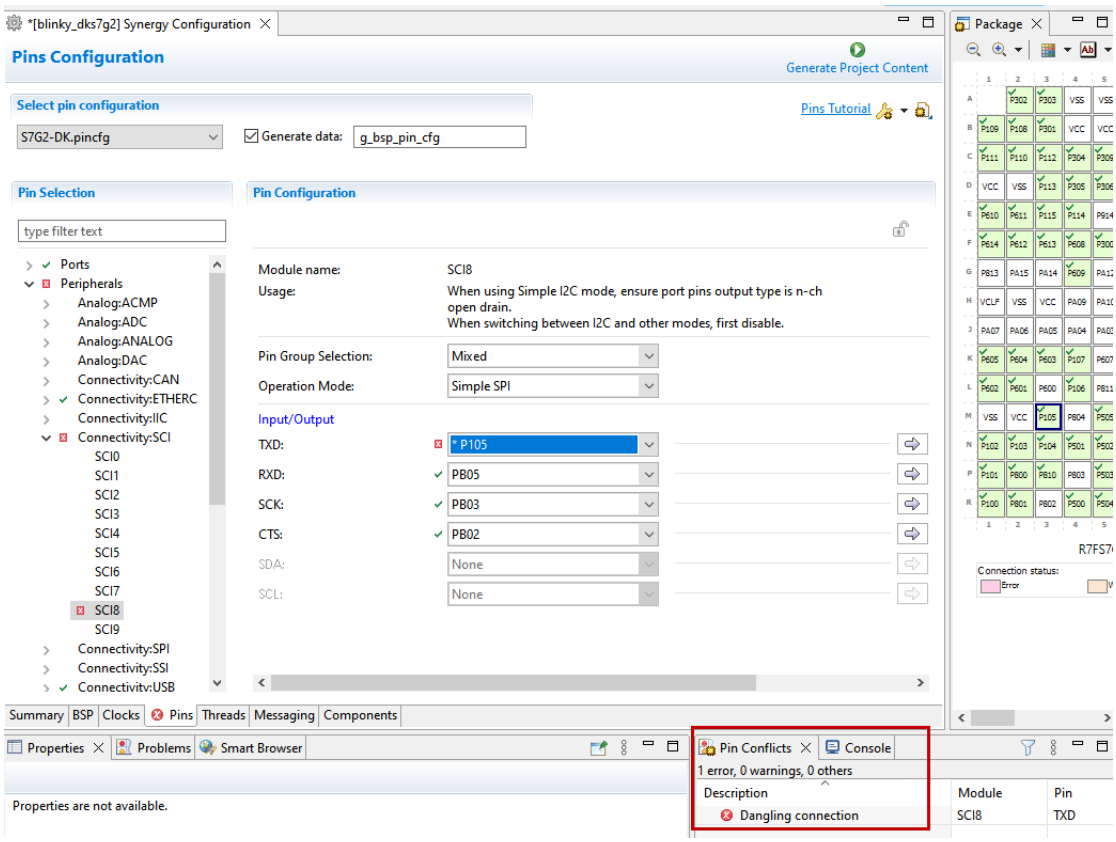


Figure 33: ISDE Pin Configurator

The pin configurator also shows a package view and the selected electrical or functional characteristics of each pin.

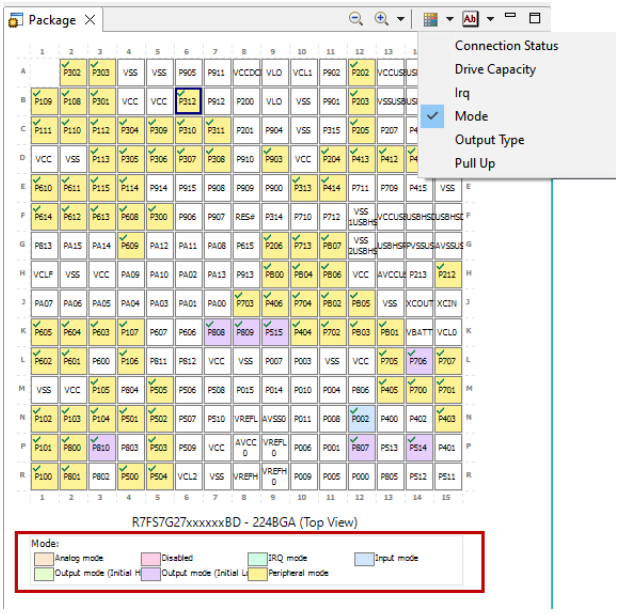


Figure 34: ISDE Pin Configurator

When you press the **Generate Project Content** button, the pin configuration contents are written to:

synergy_cfg/ssp_cfg/bsp/bsp_pin_cfg.h

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

To make it easy to share pinning information for your project, the ISDE exports your pin configuration settings to a csv format and copies the csv file to synergy_cfg/ssp_cfg/bsp/pincfg_<MCU package>.csv.

3.1.7 Adding Threads and Drivers

Every ThreadX-based Synergy Project includes at least one RTOS Thread and a stack of SSP modules running in that thread. The **Threads** tab is a graphical user interface which helps you to add the right modules to a thread and configure the properties of both the threads and the modules associated with each thread. Once you have configured the thread, the ISDE automatically generates the code reflecting your configuration choices.

For any driver or, more generally, any module that you add to a thread, the ISDE automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Threads pane, which the ISDE populates with the selected modules and module options for the selected thread. If there is more than one module that can fulfill a dependency requirement, the ISDE prompts you to choose a module from a dropdown menu.

For example, when you add the Audio Playback Framework to a thread, you also must pick either the DAC or the I2S framework for playback:

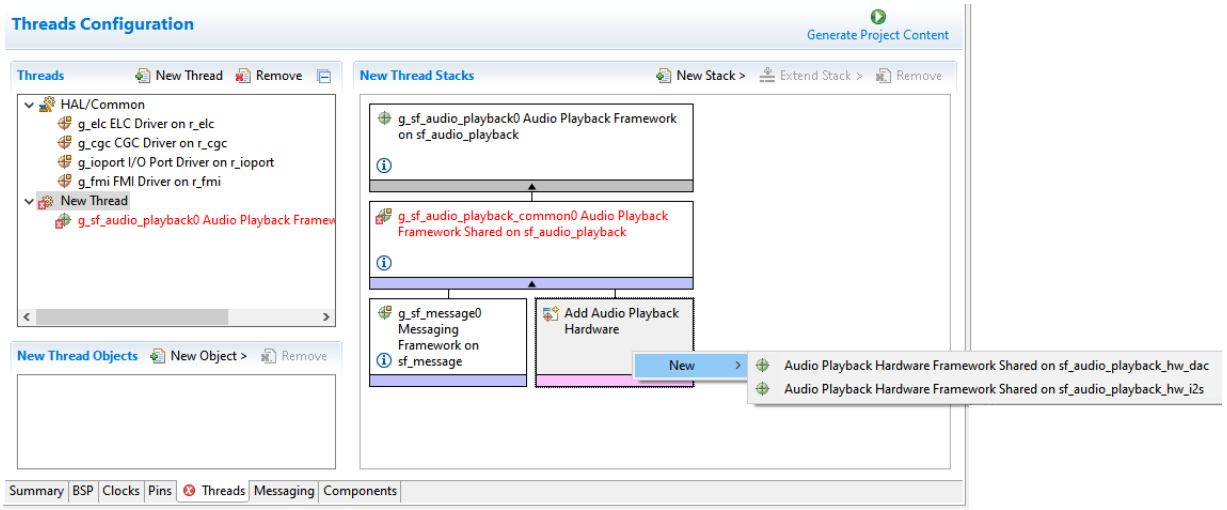


Figure 35: ISDE Project Configurator - Overview

The default view of the **Threads** tab includes a Common Thread called **HAL/Common.** This thread includes the drivers for I/O control (IOPORT), clock generation circuit (CGC), and the event link controller (ELC). The default stack is shown in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common thread are special in that the SSP only requires a single instance of each,

which the ISDE then includes in every user-defined thread by default.

In applications that do not use an RTOS or run outside of the RTOS, the HAL/Common thread becomes the default location where you can add additional drivers to your application.

For a detailed description on how to add and configure modules and threads, see the following sections:

- Adding and Configuring HAL Drivers
- Adding Drivers to a Thread and Configuring the Drivers

Once you have added a module either to HAL/Common or to a new thread, you can access the driver's configuration options in the **Properties** view. If you added thread objects, you can access the objects configuration options in the **Properties** view in the same way.

You can find details about how to configure threads here: Configuring Threads

Note

Driver and module selections and configuration options are defined in the SSP pack and can therefore change when the SSP version changes.

3.1.7.1 Adding and Configuring HAL Drivers

For applications that run outside or without the RTOS, you can add additional HAL drivers to your application using the HAL/Common thread. To add drivers, follow these steps:

1. Click on the HAL/Common icon in the **Threads** pane. The Modules pane changes to **HAL/Common Stacks**.

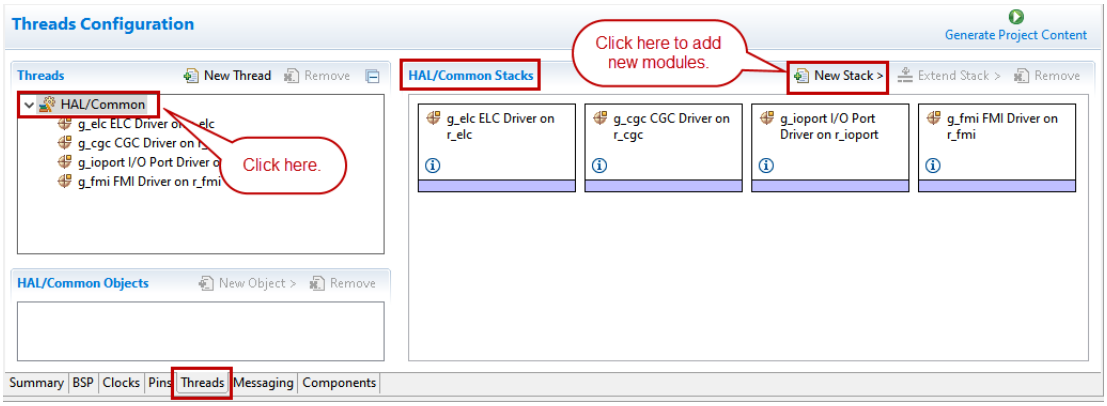


Figure 36: ISDE Project Configurator - Adding Drivers

2. Click **New Stack** to see a drop-down list of HAL level drivers available in the SSP.
3. Select a driver from the menu **New Stack > Driver**. In addition, you can select a subset of Framework modules for RTOS independent applications. All other modules can only be added to a thread when ThreadX is present.

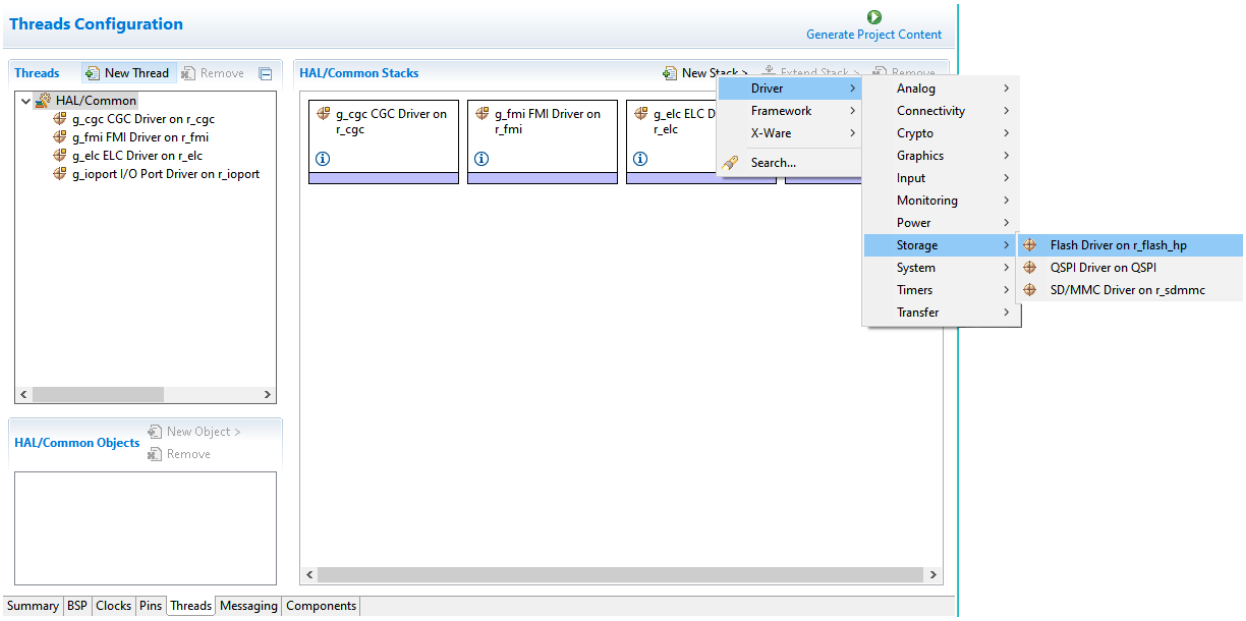


Figure 37: Select a Driver

The ISDE creates the stack for the selected driver and alerts you when the driver needs additional resources that must be enabled. In the case below, you can configure the interrupt in the **Properties** view.

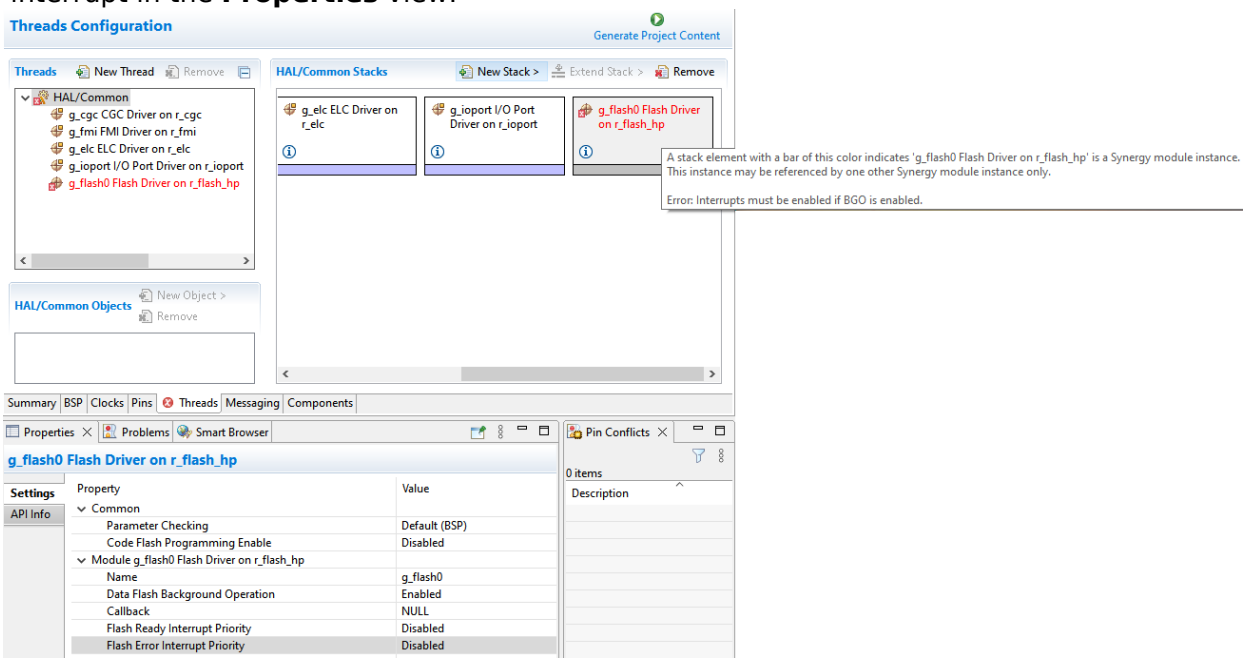


Figure 38: Dependency Checking on the Threads Tab

4. Select the driver module in the **HAL/Common Modules** pane and configure the driver properties in the **Properties** view.

The ISDE adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the synergy/ssp directory.
- The main() function and configuration structures and header files for your application as shown in the table below.

File	Contents	Overwritten by Generate Project Content?
src/synergy_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has initialized the MCU.	Yes
src/synergy_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
src/synergy_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

The configuration header files for all included modules are created or overwritten in this folder:

synergy_cfg/ssp_cfg/driver

3.1.7.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the ThreadX RTOS, you can add one or more threads, and for each thread at least one module that runs in the thread. You can select modules from either the Driver or Framework dropdown menu. To add modules to a thread, follow these steps:

1. In the **Threads** pane, click **New Thread** to add a Thread.

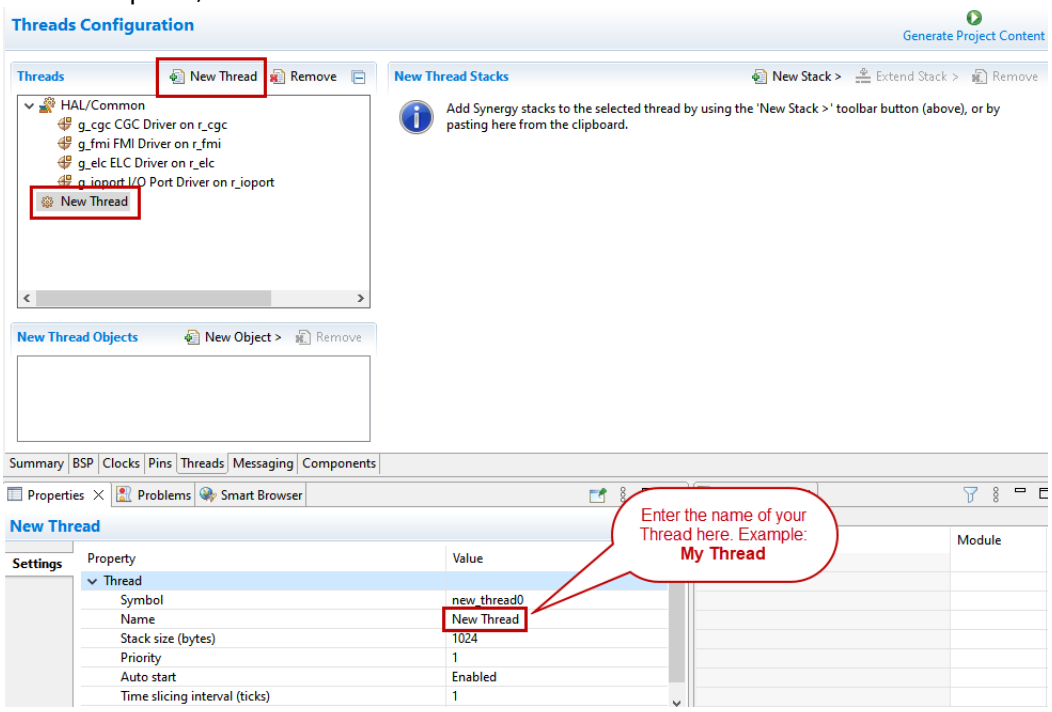


Figure 39: Adding a new RTOS Thread on the Threads Tab

2. In the **Properties** view, click on the **Name** and **Symbol** entries and enter a distinctive name and symbol for the new thread.

Note

*The ISDE updates the name of the thread stacks pane to **My Thread Stacks**.*

3. In the **My Thread Stacks** pane, click on **New** to see a list of modules and drivers. Both Framework-level Modules and HAL-level drivers can be added here.

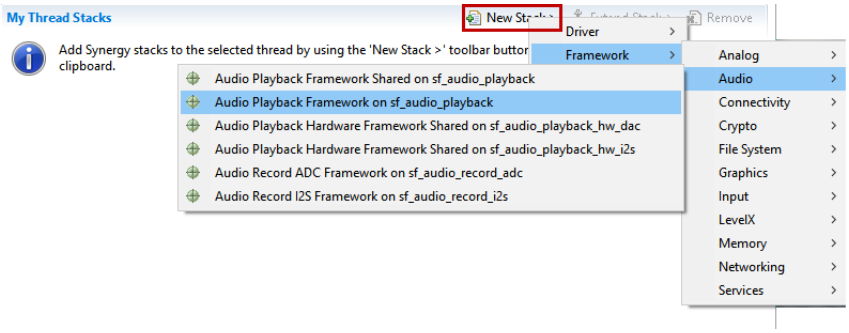


Figure 40: Adding Modules and Drivers to a Thread

4. Select a module or driver from the list.
5. If the module or driver indicates a dependency, select the missing resources.

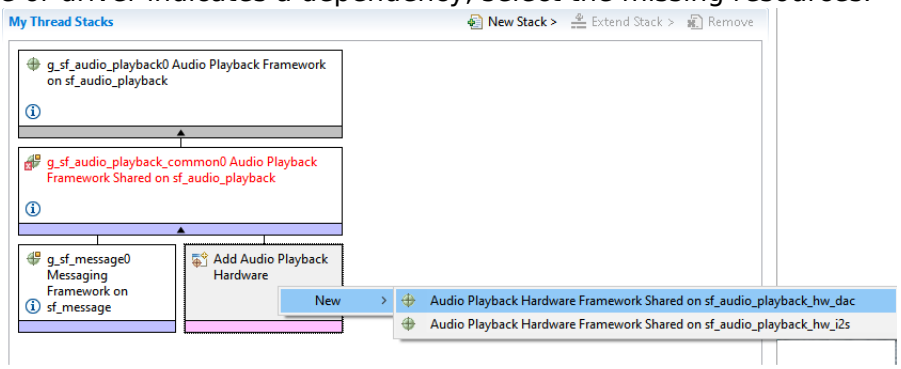


Figure 41: Identifying Module or Driver Dependencies on the Threads Tab

6. Click on the added driver and configure the driver as required by the application by updating the configuration parameters in the **Properties** view. To see the selected module or driver and be able to edit its properties, make sure the Thread containing the driver is highlighted in the **Threads** pane.

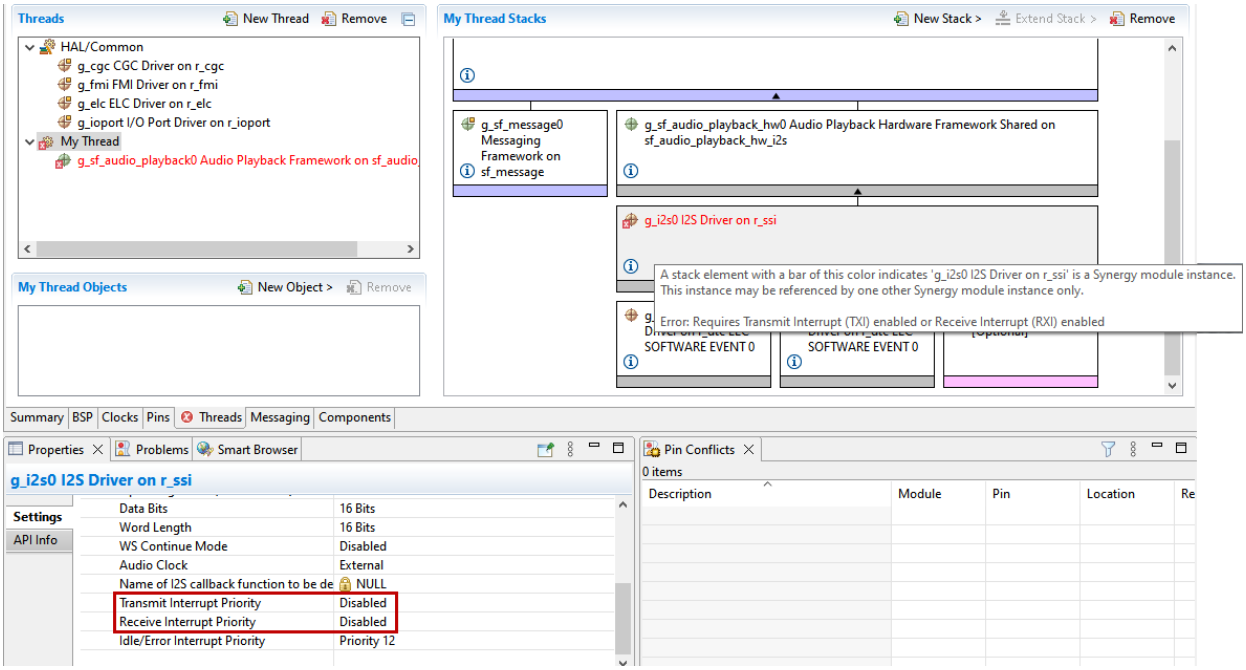


Figure 42: Configuring Module or Driver Properties

7. If needed, add another thread by clicking **New** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, the ISDE creates the files as shown in the following table:

File	Contents	Overwritten by Generate Project Content?
src/synergy_gen/main.c	Contains main() calling generated and user code. When called the BSP will have initialized the MCU.	Yes
src/synergy_gen/my_thread.c	Generated thread "my_thread" and configuration structures for modules added to this thread.	Yes
src/synergy_gen/my_thread.h	Header file for thread "my_thread"	Yes
src/synergy_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
src/synergy_gen/hal_data.h	Header file for HAL Driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No
src/my_thread_entry.c	User entry point for thread "my_thread". Add your code here.	No

The configuration header files for all included modules and drivers are created or overwritten in the following folders:

synergy_cfg/ssp_cfg/driver

synergy_cfg/ssp_cfg/framework

3.1.7.3 Configuring Threads

If the application uses the ThreadX RTOS, the **Threads** tab can be used to simplify the creation of ThreadX threads, semaphores, mutexes, and event flags.

The components of each thread can be configured from the **Properties** view as shown below.

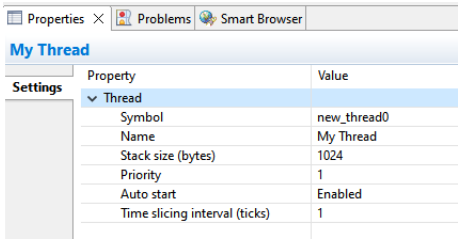


Figure 43: ISDE Thread Properties

The **Properties** view contains settings common for all Threads (**Common**) and settings for this particular thread (**Thread**).

For this thread instance, the thread's name and properties (such as priority level or stack size) can be easily configured. The ISDE checks that the entries in the property field are valid. For example, the ISDE ensures that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add ThreadX resources to a Thread, select a thread and click on **New Object** in the Thread Objects pane. The pane takes on the name of the selected thread, in this case **My Thread Objects**.

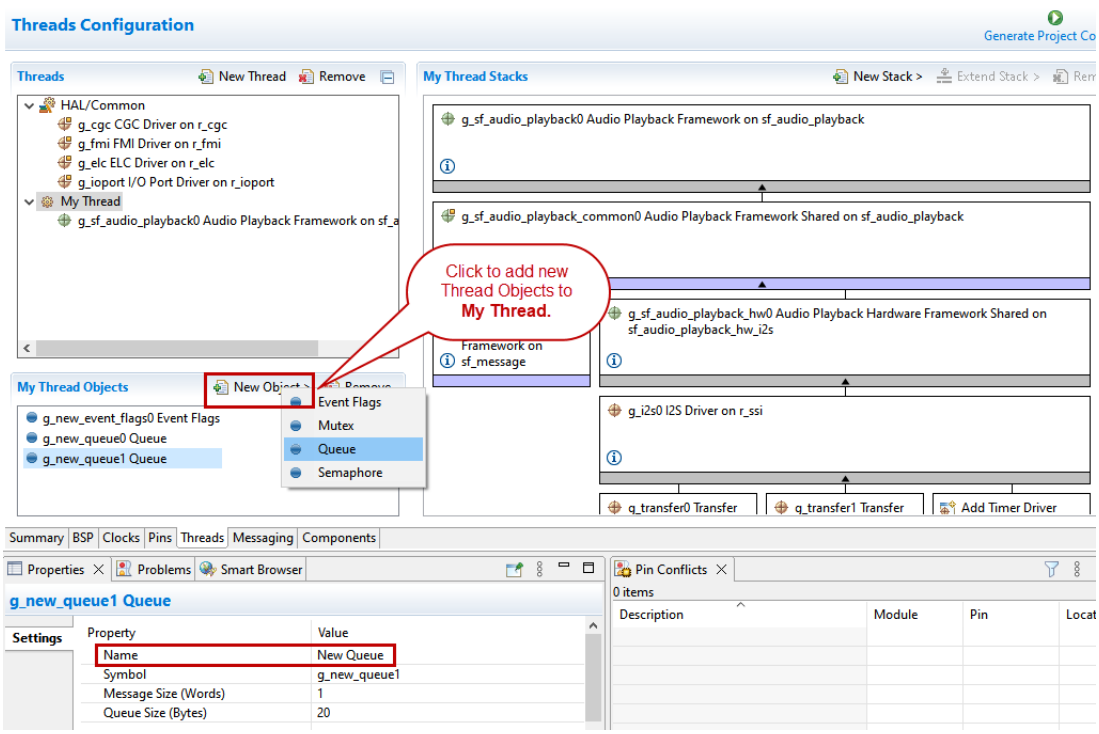


Figure 44: Configuring Thread Object Properties

Make sure to give each thread object a unique name and symbol by updating the **Name** and **Symbol** entries in the **Properties** view.

3.1.7.4 Configuring Interrupts

You can use the **Properties** view in the **Threads** tab to enable interrupts by setting the interrupt priority. Select the thread in the Threads pane to view and edit its properties.

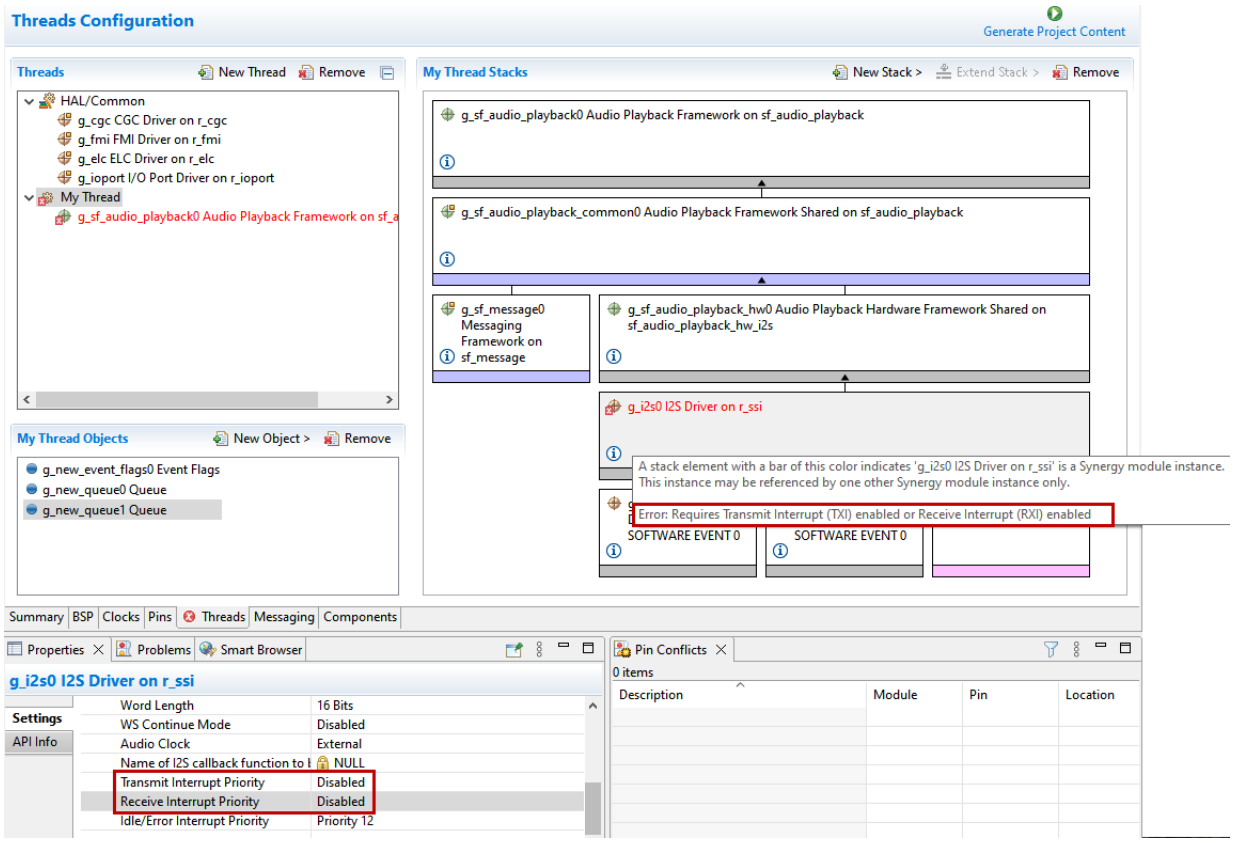


Figure 45: Configuring Interrupt on the Threads Tab

3.1.8 Configuring the SSP Messaging Framework

The Messaging Framework extends the ThreadX messaging queue functionality and is one of the most important SSP modules. It provides the mechanism for threads to communicate with each other through exchanging messages. The Messaging Framework allows threads to send (publish) or listen to (pend on) messages when preconfigured or user-configured Events happen. Any thread can publish a message with an attached Event Class that all threads subscribing to this Event class can listen to and act upon. The list of Threads that can listen to a specific Event Class is called the Subscriber List for that Event Class.

To use the Messaging Framework, you first must add one Messaging Framework instance in the **Threads** tab. You may add the Messaging Framework to any thread which is not the HAL/Common thread. All threads in your project can use this instance to communicate with each other. Some modules like the Audio Playback Framework require the Messaging Framework and add it automatically to the stack as shown below for the Audio Playback Framework. If your project includes a thread with such a module, you do not need to add another instance of the Messaging Framework even if you add more threads to your application. All threads can share one instance of the Messaging framework.

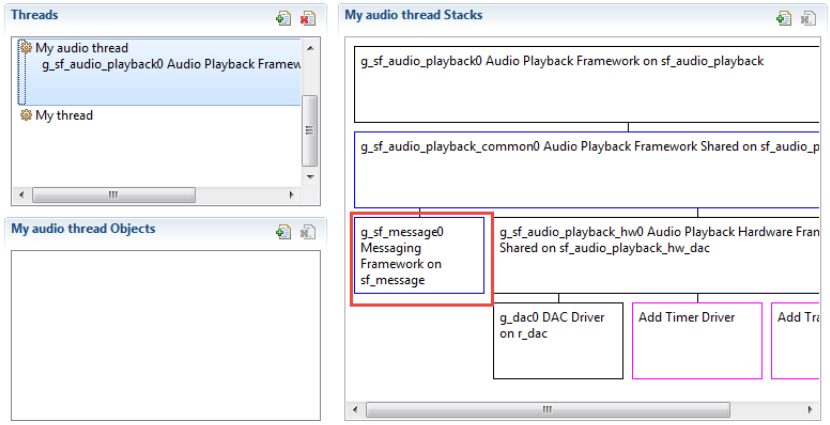


Figure 46: Adding the Messaging Module to a Thread

Once you have added a Messaging Framework instance in the **Threads** tab, you can use the **Messaging** tab to define your own event classes and events and determine which threads can listen to which event class. The SSP contains a predefined event class and events for the Audio Playback Framework module. If you have added the Audio Playback Framework module, the predefined event class and events appear in the Messaging tab as well, as shown below.

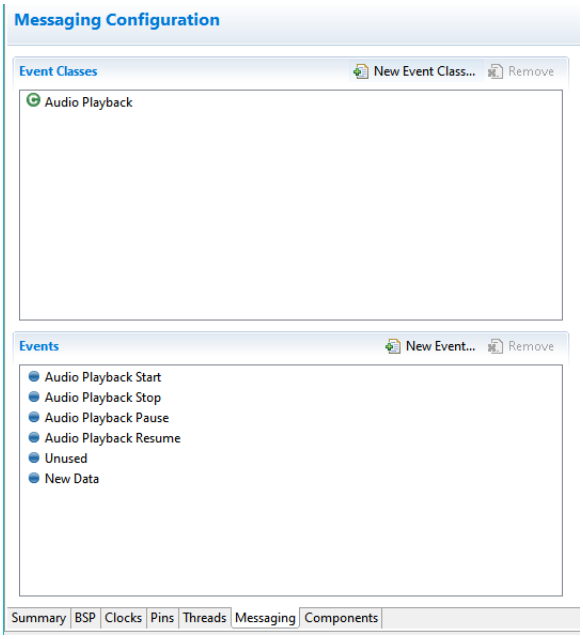


Figure 47: Audio Playback Framework Predefined Event Class

3.1.8.1 Adding an Event Class

To add your own user-defined Event Class to the messaging system, follow these steps:

1. In the **Messaging** tab, select the **Event Classes** Pane, and click the add button.
2. Enter a unique name for your event class.
3. Click **OK**.

Note

User-defined Event Classes are marked with a golden square on the upper right of the Event Class icon.

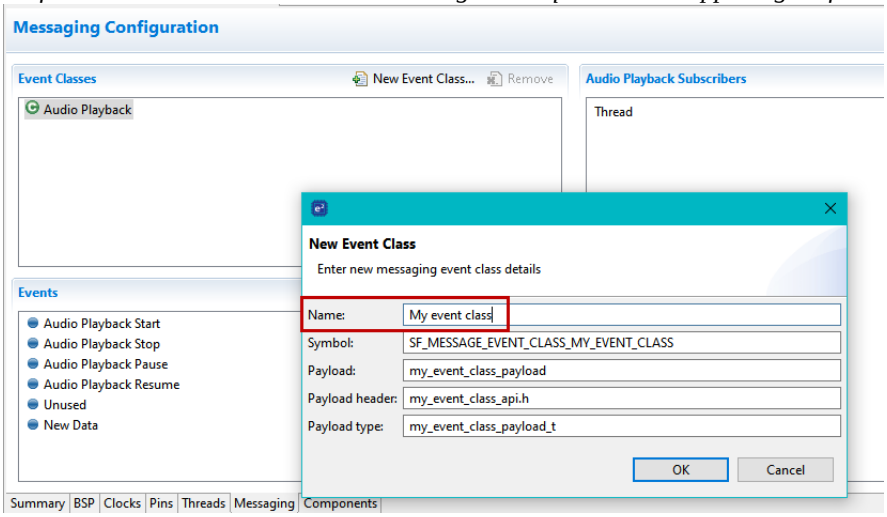


Figure 48: Messaging – Add an Event Class

3.1.8.2 Adding an Event

To add your own user-defined Event to the messaging system, follow these steps:

1. In the **Messaging** tab, select the **Event** Pane, and click the add button.
2. Enter a unique name for your event class.

3. Click **OK**.

Note

User-defined Events are marked with a golden square on the upper right of the Events icon.

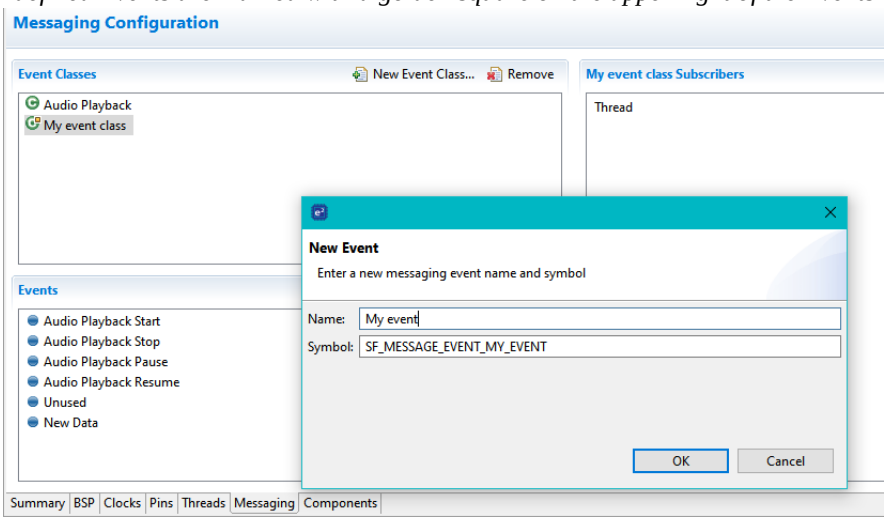


Figure 49: Messaging – Add an Event

3.1.8.3 Configuring the Messaging Subscriber List

In the Subscriber List, you select the threads that are listening to messages from the message publisher. The connection between the publishing thread and the listening thread is established through the Event Class. Therefore you define a subscriber list for each of the Event Classes in your project. All threads in the Subscriber List then can listen and act upon messages belonging to the selected Event Class.

To following assumes that you have two threads defined in the **Threads** tab, one of which uses the Audio Playback Framework:

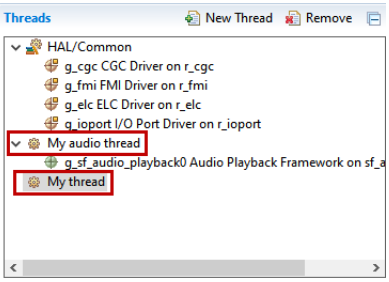


Figure 50: Messaging – Example Threads

To configure the Subscriber List for an Event Class, follow these steps:

1. In the **Messaging** tab, select the Event Classes in the Event Classes Pane.

   The Subscriber List pane takes its name from the selected Event Class.

2. Click the Add Icon.

   The New Subscriber Dialog box opens.

3. Select the Thread to add to the Subscriber List from the Thread dropdown menu.

4. Fill out the instance range by selecting Start and End.

   If you only have one instance of an Event Class, keep the Start and End values at their default value (0). See the Messaging Framework User Guide for selecting an instance range if you have more than one Event Class instance. Multiple Event Class instances can be useful in an application that uses the same Event Class multiple times for example for audio streaming on multiple channels.

5. Repeat steps 3 and 4 for each thread that you want to add to the Subscriber List for the selected Event Class.

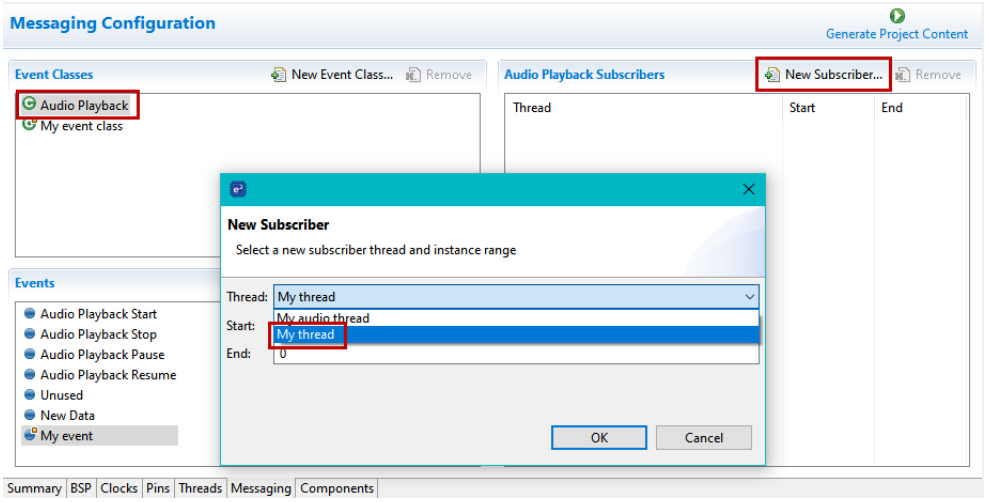


Figure 51: Messaging – Configuring the Subscriber List

3.1.8.4 Generating Files for the Messaging Framework

The ISDE generates the following files for the configured Messaging Framework when you click the Generate Project button:

File	Contents	Overwritten by Generate Project Content?
synergy_cfg/ssp_cfg/framework/sf_message_port.h	Contains the event class and event enumerations	Yes
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	Contains pointers to the event class payloads.	Yes
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	Compiler options for the Messaging Framework	Yes

3.1.9 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all Synergy projects are preselected (for example: BSP > BSP > Board-specific BSP and HAL Drivers > all > r_cgc). All modules that are necessary for the modules selected in the **Threads** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.

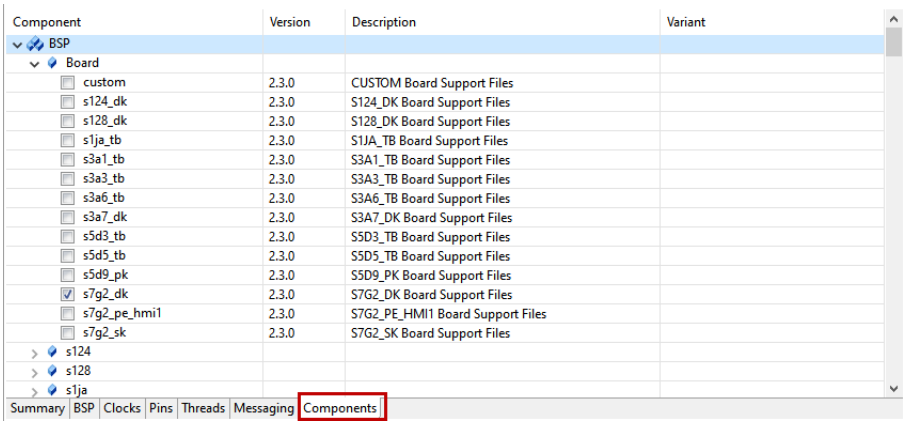


Figure 52: Components Tab

Components at the HAL and Framework layers are available as are components from Azure RTOS such as the RTOS ThreadX, file system FileX, TCP/IP networking NetX. In addition, you can select documentation to be added to a project or include complete projects.

While the components tab selects modules for a project, you must configure the modules themselves in the other tabs. Pressing the **Generate Project Content** button copies the .c and .h files for each component for a Pack file into the following folders:

synergy/ssp/inc/bsp

synergy/ssp/inc/driver

synergy/ssp/inc/framework

synergy/ssp/src/bsp

synergy/ssp/src/driver

synergy/ssp/src/framework

The ISDE also creates configuration files in the synergy_cfg/ssp_cfg folder with configuration options included from the remaining **Threads** tabs.

3.1.10 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the Threads tab, you can add the application code that calls the Modules and drivers.

Note

To check your configuration, build the project once without errors before adding any of your own application code.

3.1.10.1 RTOS-independent Applications

To write application code:

1. Add all drivers and modules in the **Threads** tab and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
2. Configure the drivers in the **Properties** view.

3. In the Project Configuration view, press the **Generate Project Content** button.
4. In the **Project Explorer** view, double-click on the src/hal_entry.c file to edit the source file.

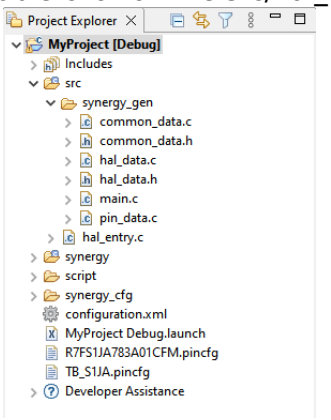


Figure 53: hal_entry.c

Note

All configuration structures necessary for the driver to be called in the application are initialized in src/synergy_gen/hal_data.c.
*Do not modify the files in the directory src/synergy_gen. These files are overwritten every time you push the **Generate Project Content** button.*

5. Add your application code here:

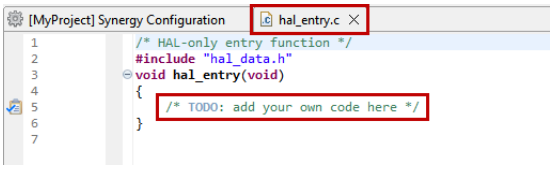


Figure 54: Adding User Code to hal_entry.c

6. Build the project without errors by clicking on **Project > Build Project**.

The following tutorial shows how execute the steps above and add application code: Tutorial: Using HAL Drivers - Programming the WDT

The WDT example is a HAL level application which does not use an RTOS. The user guides for each module also include basic application code that you can add to hal_entry.c.

3.1.10.2 ThreadX Applications

To write RTOS-aware application code using ThreadX, follow these steps:

1. Add a thread using the **Threads** tab.
2. Provide a unique name for the thread in the **Properties** view for this thread.
3. Configure all drivers and resources for this thread and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
4. Configure the thread objects.
5. Provide unique names for each thread object in the **Properties** view for each object.
6. Add more threads if needed and repeat steps 1 to 5.
7. In the **Synergy Project Editor**, press the **Generate Project Content** button.
8. In the **Project Explorer** view, double-click on the src/my_thread_1_entry.c file to edit the source file.

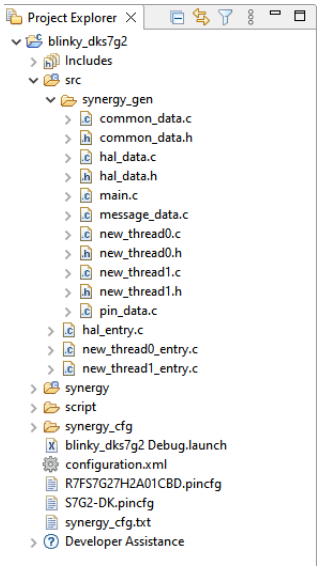


Figure 55: ISDE Generated Files for an RTOS Application

Note

All configuration structures necessary for the driver to be called in the application are initialized in synergy_gen/my_thread_1.c and my_thread_2.c
Do not modify the files in the directory src/synergy_gen. These files are overwritten every time you push the ***Generate Project Content*** *button.*

9. Add your application code here:

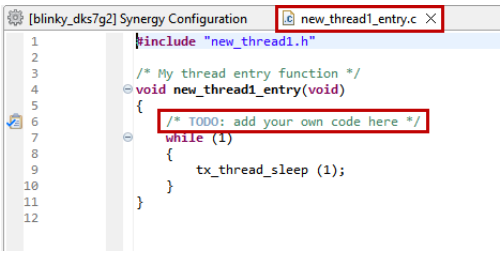


Figure 56: Adding User Code to my_thread_1.entry

10. Repeat steps 1 to 9 for the next thread.
11. Build your project without errors by clicking on **Project > Build Project**.

3.1.11 Debugging the Project

Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

1. click **Run > Debug Configurations**.

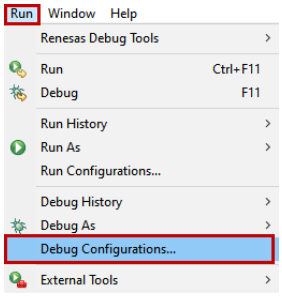


Figure 57: Invoking the Debug Configurations Dialog

2. In the **Debug Configurations** view, click on your project listed as **MyProject Debug**.

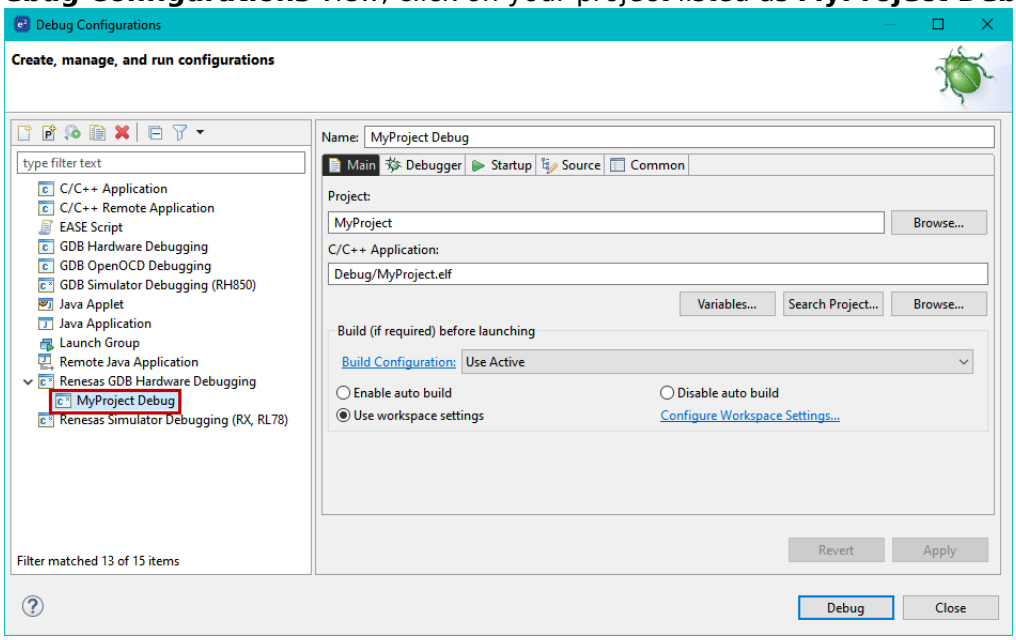


Figure 58: Debug Configuration

3. Connect the board to your PC via either a standalone Segger J-Link debugger or a Segger J-Link On-Board (included on all Synergy DKs and SKs) and click **Debug**.

Note

For details on using J-Link and connecting the board to the PC, see the Quick Start Guide included in the Synergy Kit.

3.1.12 Using TraceX with a Synergy Project

Precondition

Before you can use TraceX with your Synergy Project, you must download the TraceX executable file from the Microsoft Store.

TraceX™ is a host-based analysis tool that provides a graphical view of real-time system events. TraceX collects data on the target device and displays the data for inspection and analysis. A TraceX version for Synergy devices is available for downloading from the Microsoft Store.

To use TraceX, do the following:

1. In e^2 studio, add the ThreadX source code to your project by going to the **Threads** tab, clicking the **New Stack** button in the **Stacks** pane, and selecting **X-Ware > ThreadX > ThreadX Source**.

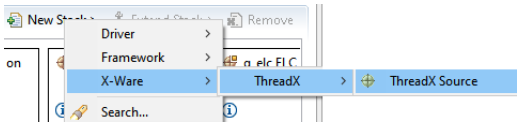


Figure 59: Add TraceX to your Source

2. Enable TraceX in the Properties Window of the Thread using the **Threads** tab. Keep the default name for the TraceX buffer as g_tx_trace_buffer.

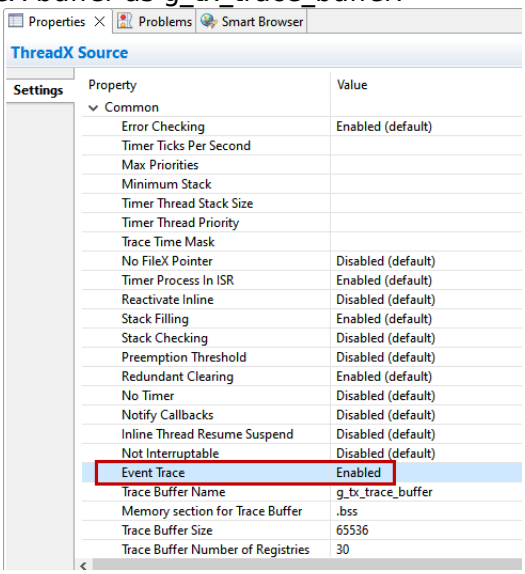


Figure 60: ISDE TraceX Configuration

3. Set the path to the TraceX application in **Window > Preferences > Renesas > TraceX**

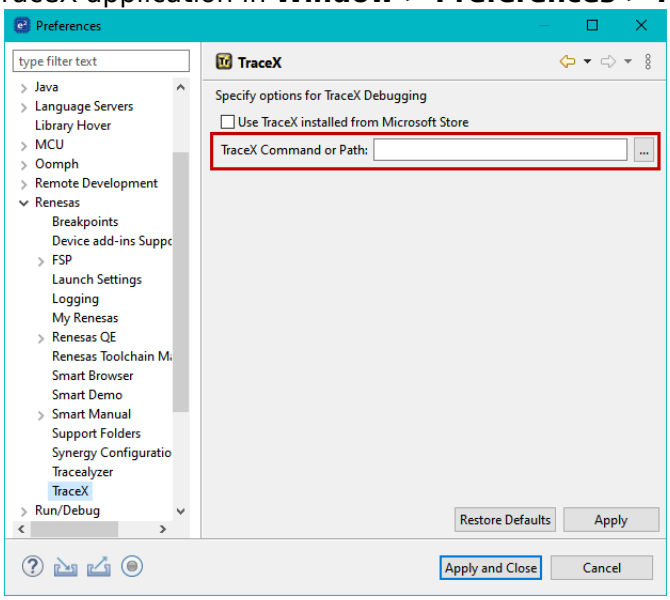


Figure 61: ISDE TraceX Path

4. Build your project (**Project > Build All**).

5. Connect your Synergy target board.
6. Start a debug session (**Run > Debug**)
7. In **Run > Renesas Debug Tools > TraceX**, select **Launch TraceX Debugging**.

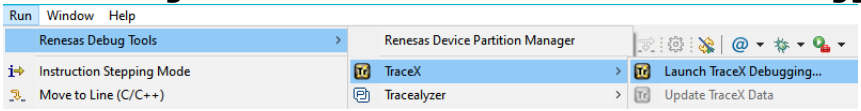


Figure 62: ISDE TraceX Launch

8. In the TraceX Debugging window, set **Buffer Start Address** to &g_tx_trace_buffer.

   In the TraceX Debugging window, set **Buffer Size (bytes)** to the buffer size selected in the Properties Window in step 2. The default is 65536.

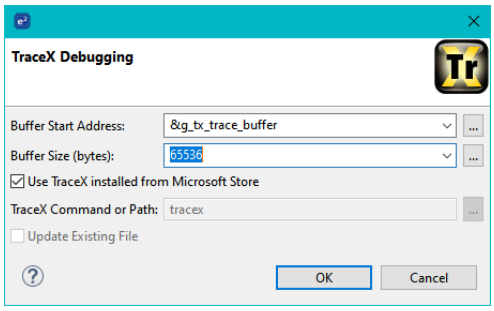


Figure 63: ISDE TraceX Debug

9. Click **OK**.
10. Run your code (**Run > Resume**) to collect TraceX data.
11. Suspend execution of your code (**Run > Suspend**).
12. Observe the collected data in TraceX.

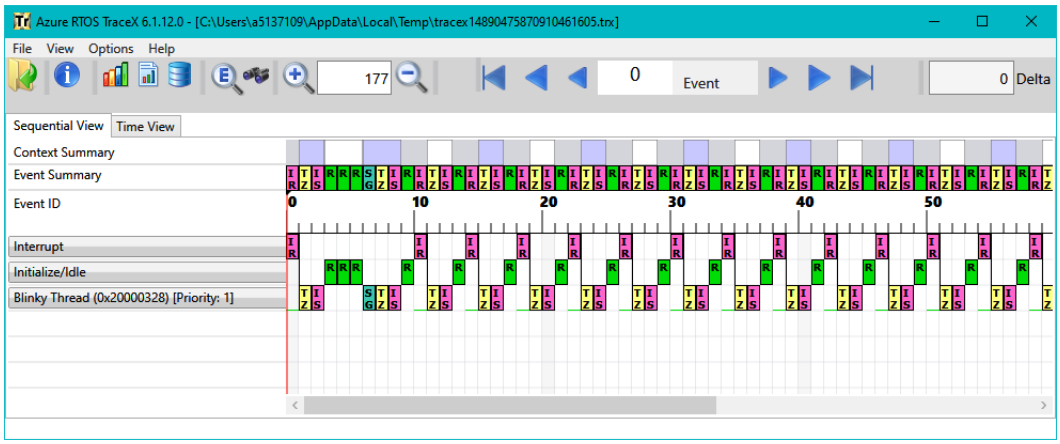


Figure 64: TraceX Collected Data

13. To collect further TraceX data:
    - Resume execution of your code
    - Suspend execution of your code
    - Click **Run > TraceX** and select **Update TraceX Data**.

You can find more information on using TraceX on the Renesas TraceX webpage

https://www.renesas.com/synergy/tracex.

3.1.13 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within the ISDE through the menu **Project > Renesas C/C++ Project Settings** when the project is selected. The following screenshot shows the settings dialog for the GNU ARM toolchain. This dialog will look slightly different depending upon the toolchain being used.

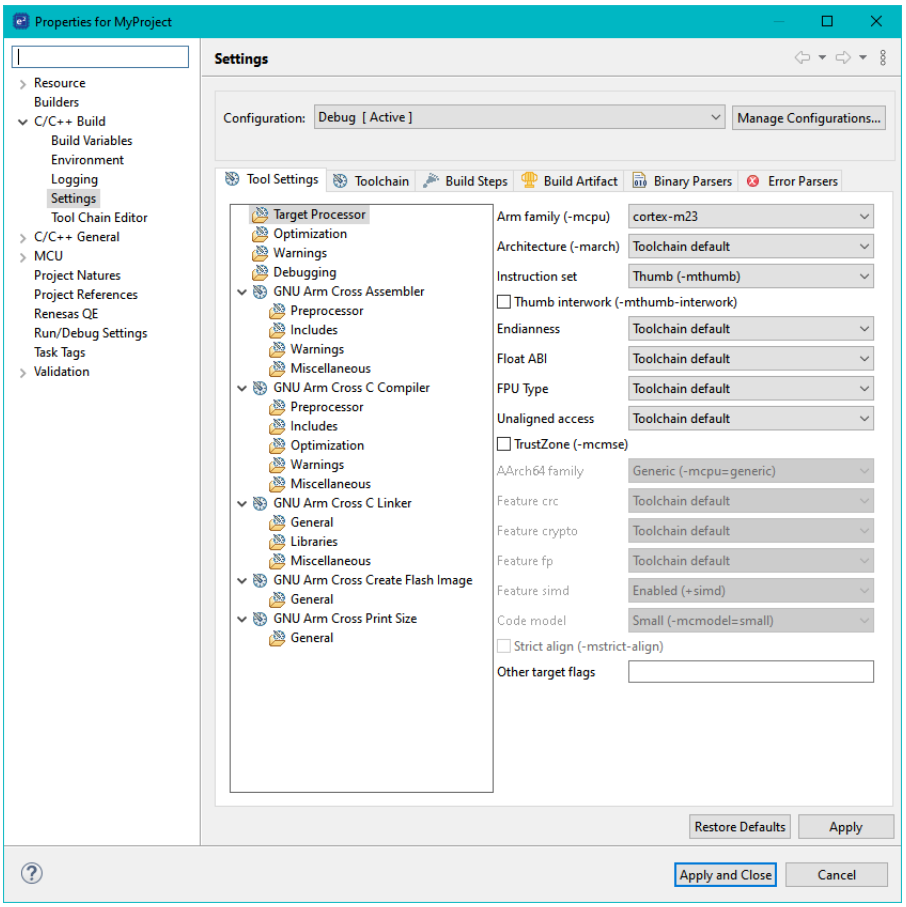


Figure 65: ISDE Project Toolchain Settings

The scope for the settings is project scope which means that the settings are valid only for the project being modified.

The settings for the linker which control the location of the various memory sections are contained in a script file specific for the device being used. This script file is included in the project when it is created and is found in the script folder (for example, /script/S7G2.ld).

3.1.14 e2 studio ISDE Usage Notes

3.1.14.1 Including ThreadX sources

You can use the **Theads** tab to include ThreadX source code in your project as follows:

1. Click on the HAL/Common icon in the **Threads** pane. The Modules pane changes to **HAL/Common Modules**.
2. Select **New Stack** > **X-Ware** > **ThreadX** > **ThreadX Source**.
3. Check the **Properties** window to configure the TheadX RTOS properties.
4. Click **Generate Project**.

The e^2 studio ISDE extracts the ThreadX source code into the following directory: synergy/ssp/src/framework/el/tx

Note

Extracting the ThreadX sources increases the compile time for your project.

3.1.14.2 Using Synergy Developer Assistance

This section describes a new feature, "Synergy Developer Assistance" included in e^2 studio.

Developer Assistance Node

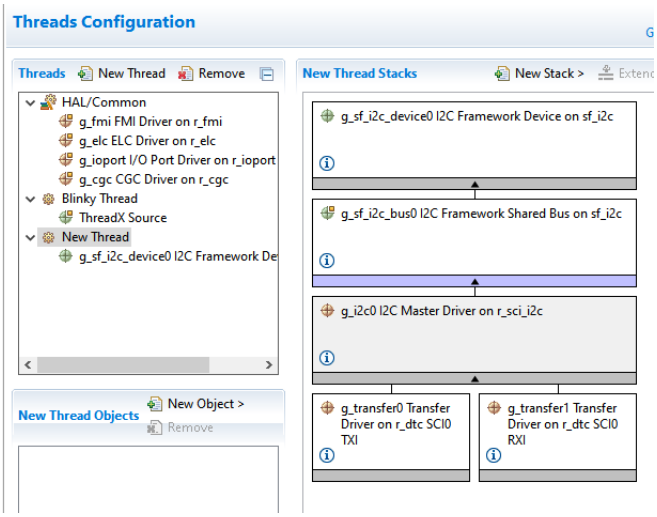
A new node **Developer Assistance** is now available in the project explorer. When expanded, the Developer Assistance tree shows you the threads and module stacks and their respective API information of the saved Synergy configuration.

The Developer Assistance node is available regardless of whether the Synergy Configuration Editor is open. This allows a user to easily consult the Developer Assistance for a project's stack modules while application code is being written. It assists by providing code templates and an autocompletion tool.

Configure Threads

Create a new project. In the Synergy configurator thread's tab create a thread and add a new stack.

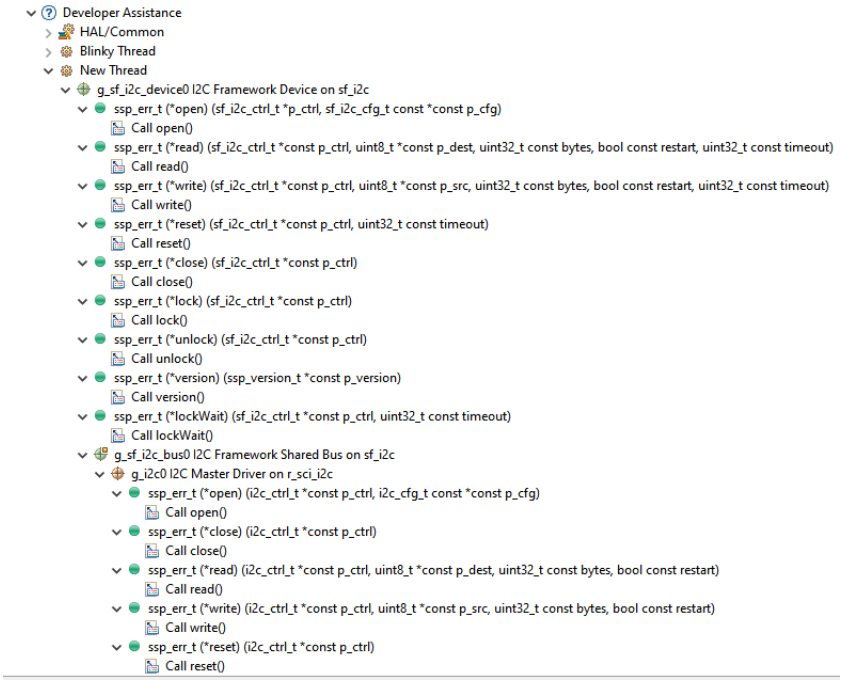
For example, Framework Device on sf_i2c:



Project Explorer View

Once the project is saved, expand the **Developer Assistance** node. Notice that the node is updated

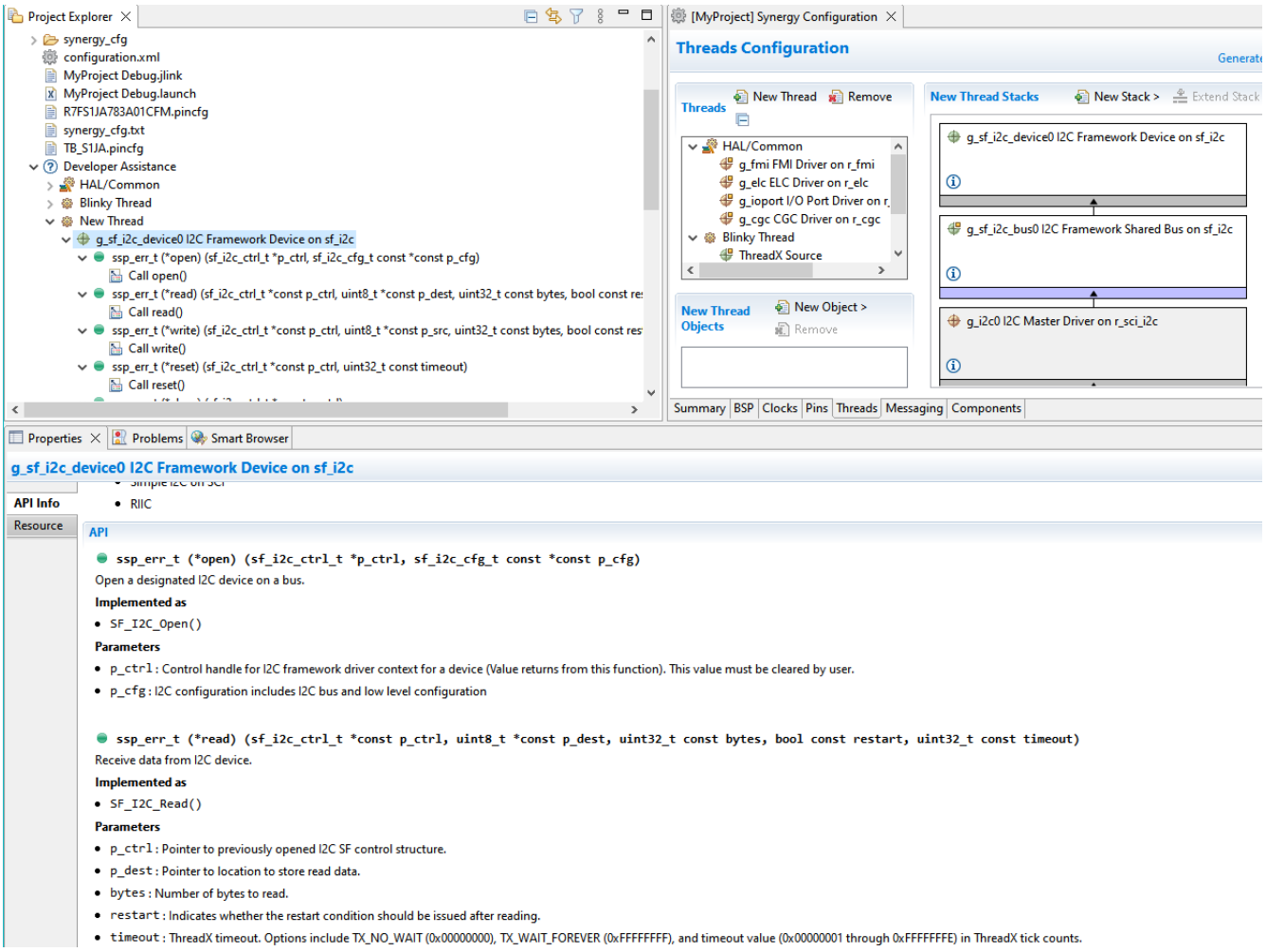
with the new thread **New Thread** Information. Clicking on the module instance under the thread will expand all the child nodes and show the API information of the modules.



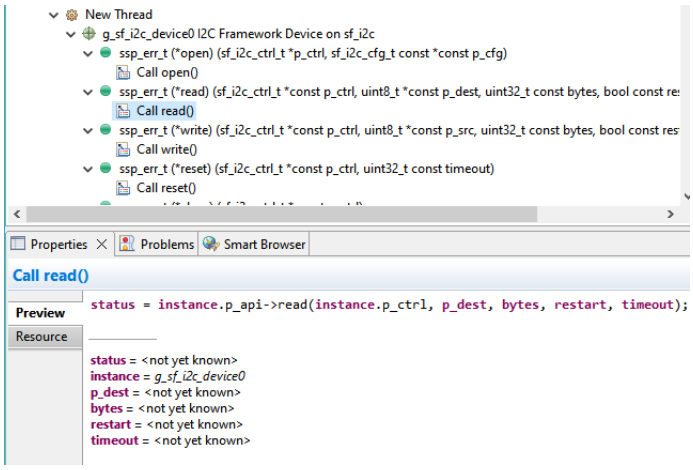
Properties View

Selecting the module instance in the thread tab or in the project explorer will show the module and module API information in the API info tab of properties view.

Tip: If the Properties view is not present in the current e^2 studio perspective, you can open it by selecting **Window** -> **Show View** -> **Properties**

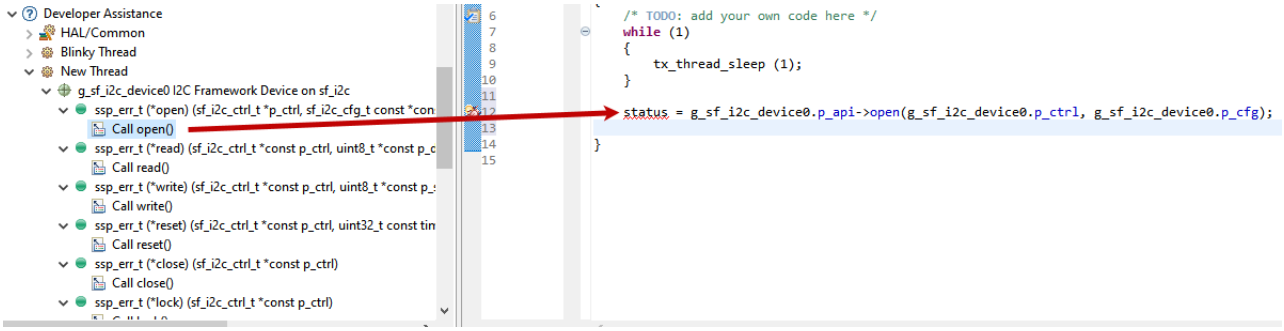


Selecting the code template node shows the preview of the function in the **Properties** view



Drag and drop to source File

The API function code templates or callbacks can be dragged and dropped to the source file. This will autofill the parameters and return type of the function



Please also refer to **Help**-> **Help Contents** -> **Synergy Contents** -> **Synergy Developer Assistance** page in e^2 studio for any further information.

3.2 Tutorial: Your First Synergy Project - Blinky

3.2.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Synergy Platform by moving through the steps of creating a simple application using e2 studio and running that application on a Synergy board.

3.2.2 What Does Blinky Do?

The application used in this tutorial is Blinky, traditionally the first program run in a new embedded development environment.

Blinky is the "Hello World" of microcontrollers. If the LED blinks you know that:

- The toolchain is setup correctly and builds a working executable image for your chip.
- The debugger has installed with working drivers and is properly connected to the board.
- The board is powered up and its jumper and switch settings are probably correct.
- The microcontroller is alive, the clocks are running, and the memory is initialized.

The Blinky example application used in this tutorial is designed to run the same way on all boards offered by Renesas that hold the Synergy microcontroller. The code in Blinky is completely board independent. It does the work by calling into the BSP (board support package) for the particular board it is running on. This works because:

- Every board has at least one LED connected to a GPIO pin.
- That one LED is always labeled LED1 on the silk screen.
- Every BSP supports an API that returns a list of LEDs on a board, and their port and pin

assignments.

3.2.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e^2 studio
- Synergy Software Package
- A Synergy board kit

3.2.4 Create a New Project for Blinky

The creation and configuration of a Synergy project is the first step in the creation of an application. The base SSP pack includes a pre-written Blinky example application that is simple and works on all Renesas Synergy boards.

Note

The e^2 studio screens shown in this manual are examples. Some details may differ between different releases of the e^2 studio ISDE and the SSP.

Follow these steps to create a Synergy project:

1. In e^2 studio ISDE, click **File > New > Synergy C/C++ Project**, select **Renesas Synergy C Executable Project**, and click **Next**.
2. Assign a name to this new project. Blinky is a good name to use for this tutorial.

3. Click **Next**. The Project Configuration window shows your selection.

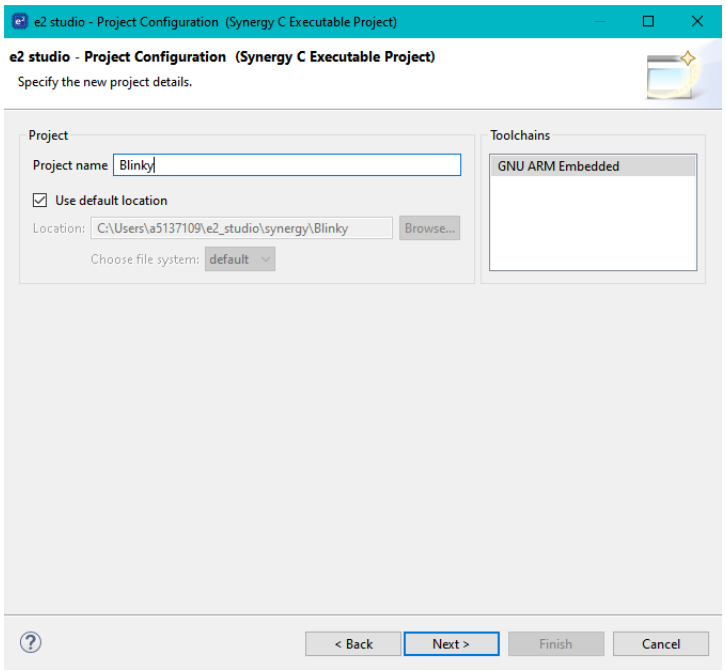


Figure 66: e2 studio ISDE Project Configuration Window (Part 1)

4. Select the board support package by selecting the name of your board from the **Device Selection** drop-down list and click **Next**.

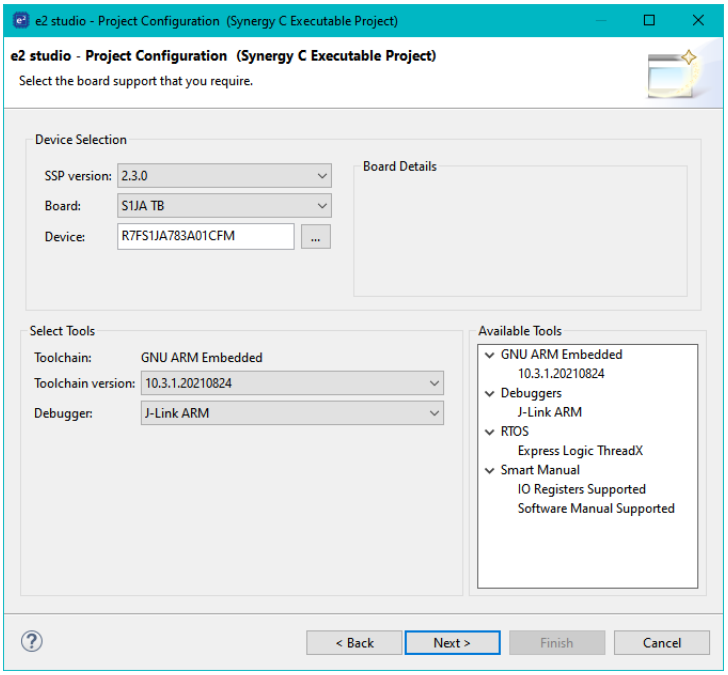


Figure 67: e2 studio ISDE Project Configuration Window (Part 2)

5. Select the Blinky template for your board and click **Finish**.

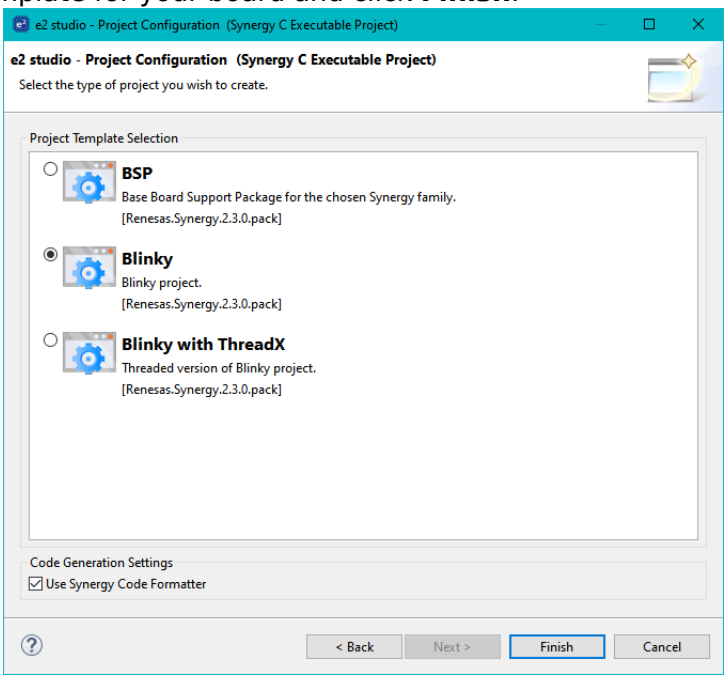


Figure 68: e2 studio ISDE Project Configuration Window (Part 3)

Once the project has been created, the name of the project will show up in the Project Explorer window of the ISDE. Now press the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

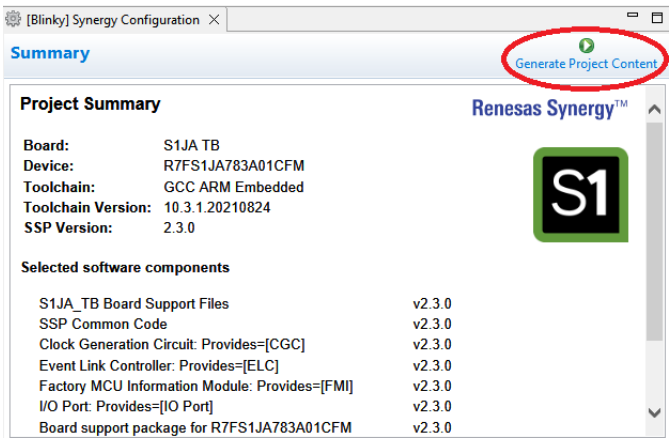


Figure 69: e2 studio ISDE Project Configuration Tab

Your new project is now created, configured, and ready to build.

3.2.4.1 Details about the Blinky Configuration

The **Generate Project Content** button creates configuration header files, copies source files from templates, and generally configures the project based on the state of the **Project Configuration** screen.

For example, if you check a box next to a module in the **Components** tab and press the **Generate Project Content** button all the files necessary for the inclusion of that module into the project will be copied or created. If that same check box is then unchecked those files will be deleted.

3.2.4.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by the ISDE for the Blinky application. The ISDE clock configuration tab (see Configuring Clocks) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file (see BSP Clock Configuration).

3.2.4.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by the ISDE for the Blinky application. The ISDE pin configuration tab shows the pin configuration for the Blinky application (see Configuring Pins). The Blinky pin configuration is stored in the BSP configuration file (see BSP Pin Configuration).

3.2.4.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the ISDE Component:

- r_cgc
- r_elc
- r_fmi
- r_ioport

To see the configuration parameters for any of the components, check the Properties tab in the **HAL** window for the respective driver (see Adding and Configuring HAL Drivers).

3.2.4.5 Where is main()?

The main function is located in < project >/src/synergy_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files Adding and Configuring HAL Drivers .

3.2.4.6 Blinky Example Code

The blinky application is stored in the hal_entry.c file. This file is generated by the ISDE when you select the Blinky Project template and is located in the project's src/ folder.

The application performs the following steps:

1. Get the LED information for the selected board by calling the BSP HAL function R_BSP_LedsGet().
2. Define the output level HIGH for the GPIO pins controlling the LEDs for the selected board.
3. Get the selected system clock speed and scale down the clock, so the LED toggling can be observed.
4. Toggle the LED by writing to the GPIO pin with

```
g_ioport.p_api->pinWrite()
```

3.2.5 Build the Blinky Project

Highlight your new project in the **Project Explorer** window and build it.

There are three ways to build a project:

a. Click on Project in the menu bar and select Build Project.

b. Click on the hammer icon.

c. Right-click on the project and select Build Project.

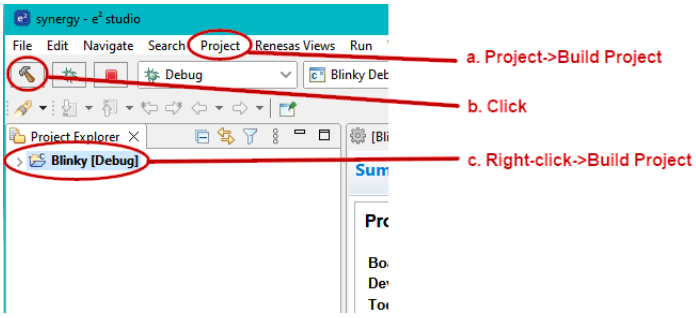


Figure 70: e2 studio ISDE Project Explorer Window

Once the build is complete a message is displayed in the build **Console** window that displays the final image file name and section sizes in that image.

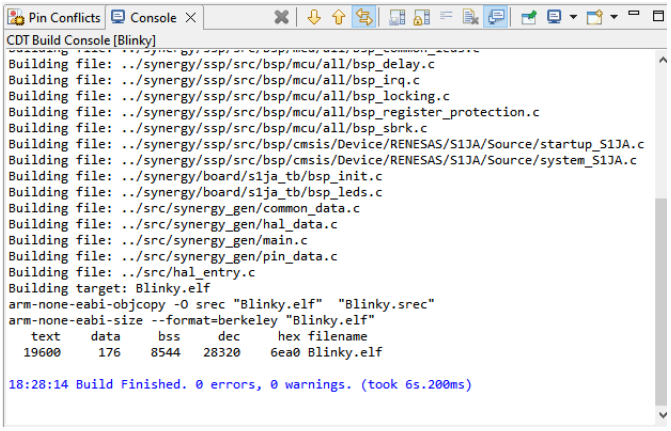


Figure 71: e2 studio ISDE Project Build Console

3.2.6 Debug the Blinky Project

3.2.6.1 Debug prerequisites

To debug the project on a board, you need:

- The board to be connected to the ISDE
- The debugger to be configured to talk to the board
- The application to be programmed to the microcontroller

Applications run from the internal flash of your microcontroller. To run or debug the application, the application must first be programmed to the microcontroller's flash. There are two ways to do this:

- JTAG debugger
- Built-in boot-loader via UART or USB

Some boards have an on-board JTAG debugger and others require an external JTAG debugger connected to a header on the board.

Refer to your board's user manual to learn how to connect the JTAG debugger to your ISDE.

3.2.6.2 Debug steps

To debug the Blinky application, follow these steps:

1. Configure the debugger for your project by clicking **Run > Debugger Configurations** ...

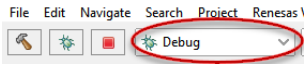


Figure 72: e2 studio ISDE Debug Icon

... or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configurations** ...

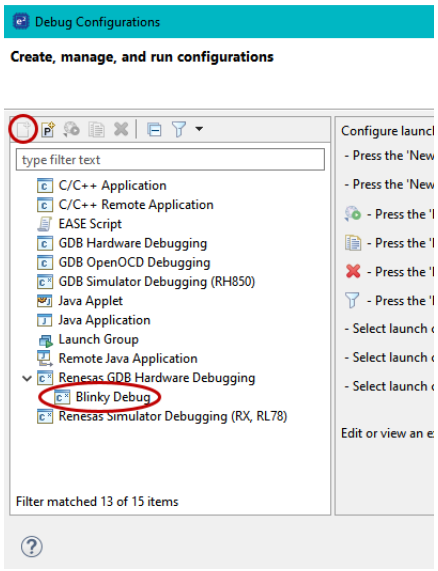


Figure 73: e2 studio ISDE Debugger Configurations Window

2. Select your debugger configuration in the window. If it is not visible then it must be created by clicking the **New** icon in the top left corner of the window. Once selected, the **Debug Configuration** window displays the Debug configuration for your Blinky project.

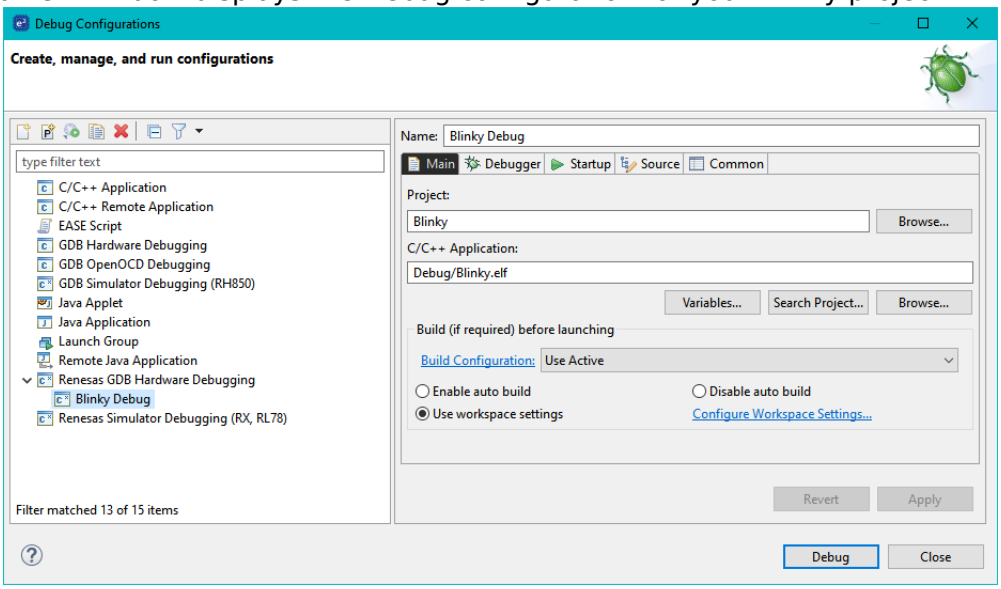


Figure 74: e2 studio ISDE Debugger Configurations Window with Blinky Project

3. Press **Debug** to begin debugging the application.

3.2.6.3 Details about the Debug Process

In debug mode, the ISDE executes the following tasks:

1. Downloading the application image to the microcontroller and programming the image to the internal flash memory.
2. Setting a breakpoint at main().
3. Setting the stack pointer register to the stack.
4. Loading the program counter register with the address of the reset vector.

5. Displaying the startup code where the program counter points to.

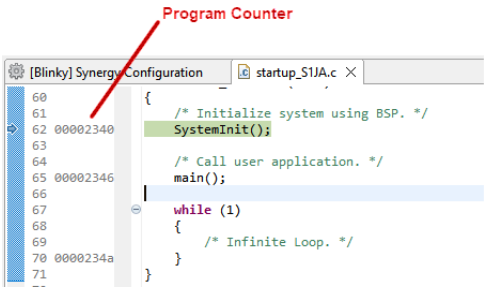


Figure 75: e2 studio ISDE Debugger Memory Window

3.2.7 Run the Blinky Project

While in Debug mode, click **Run > Resume** or click on the Play icon twice.

Figure 76: e2 studio ISDE Debugger Play Icon

The LED on the board marked LED1 should now be blinking.

3.3 Tutorial: Using HAL Drivers - Programming the WDT

3.3.1 Application WDT

This application uses the WDT Interface implemented by the WDT HAL Driver WDT. This document describes how to use the ISDE and SSP to create an application for the Synergy MCU Watchdog Timer (WDT) peripheral. This application makes use of the following SSP modules:

- Board Support Package (Board Support Package)
- CGC (Clock Generation Circuit)
- WDT (Watchdog Timer)
- IOPORT (GPIO)

3.3.2 Creating a WDT Application Using the Synergy SSP and ISDE

3.3.2.1 Using the SSP and the e2 studio ISDE

The Synergy Software Package (SSP) from Renesas provides a complete driver library for developing Synergy applications. The SSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers and higher level Framework applications for the developer to use to create applications. The SSP is integrated into the Renesas e2 studio Integrated Solution Development Environment (ISDE) based on eclipse providing build (editor, compiler and linker) and debug phases

with an extended GNU Debug (GDB) interface.

3.3.2.2 The WDT Application

The flowchart for the WDT application is shown below.

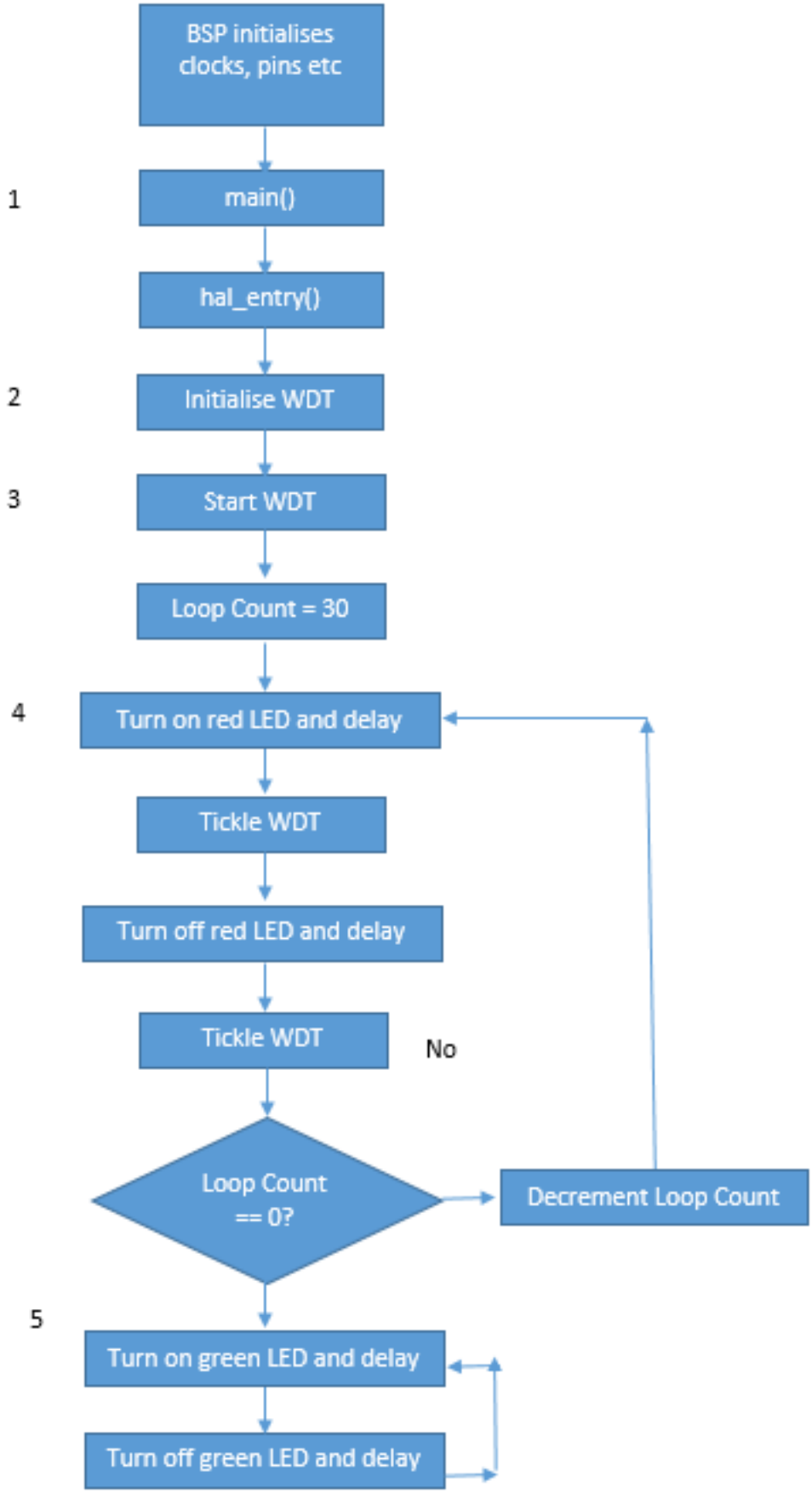


Figure 77: WDT Application Flow Diagram

3.3.2.3 WDT Application flow

These are the main parts of the WDT application:

1. main() calls hal_entry(). The function hal_entry() is created by the SSP with a placeholder for user code. The code for the WDT will be added to this function.
2. Initialize the WDT, but do not start it.
3. Start the WDT by refreshing it.
4. The red LED is flashed 30 times and refreshes the watchdog each time the LED state is changed.
5. Flash the green LED but DO NOT refresh the watchdog. After the timeout period of the watchdog the device will reset which can be observed by the flashing red LED again as the sequence repeats.

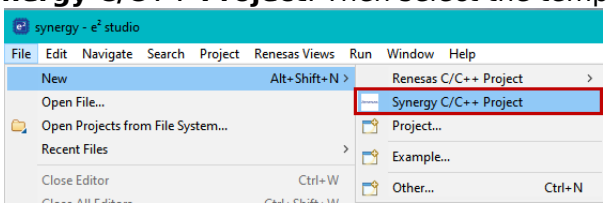
3.3.3 Creating the Project with the ISDE

Start the ISDE and choose a workspace folder in the Workspace Launcher. Configure a new Synergy project as follows.

Note

The e² studio screens shown in this manual are examples. Some details may differ between different releases of the e² studio ISDE and the SSP.

1. Select **File > New > Synergy C/C++ Project**. Then select the template for the project.



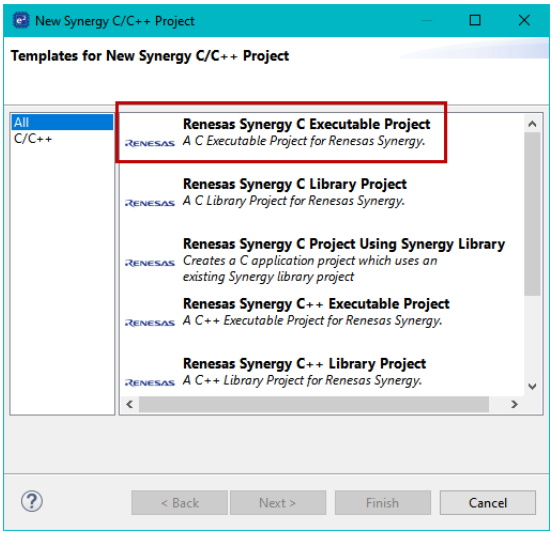


Figure 78: Creating a New Project

2. In the ISDE Project **Configuration (Synergy Project)** window, enter a project name (for

example, WDT_Application). Select the toolchain. If you want to choose a new location for the project, deselect **Use default location**. Click **Next**.

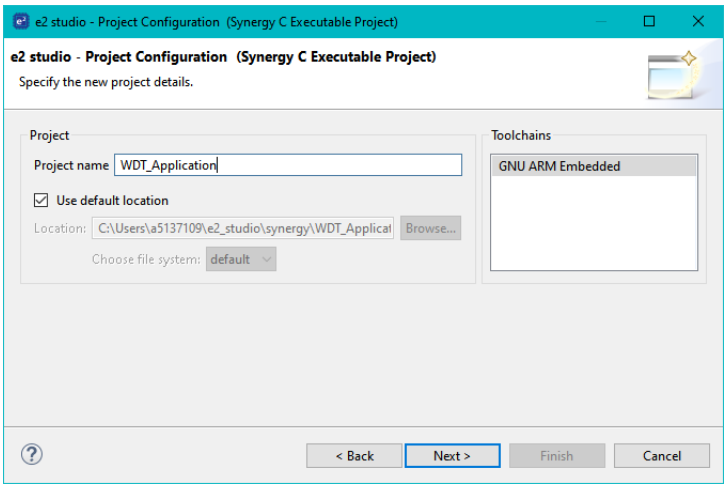


Figure 79: Project Configuration (Part 1)

3. This application runs on the Synergy S7G2 based DK-S7G2 board. So, for the **Board** select **S7G2 DK**.

   This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **J-Link ARM** as the **Debugger**. No **RTOS** is being used in this application but it can be left at the default. Click **Next** to configure the project.

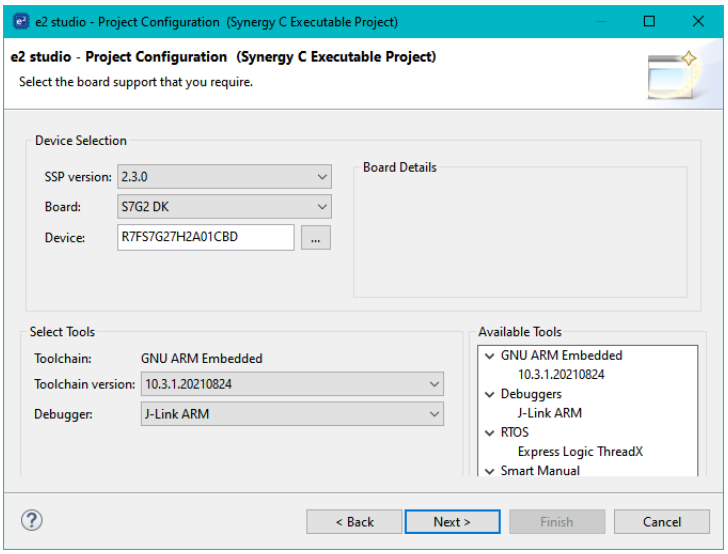


Figure 80: Project Configuration (Part 2)

The project template is now selected. As no RTOS is required select **BSP**.

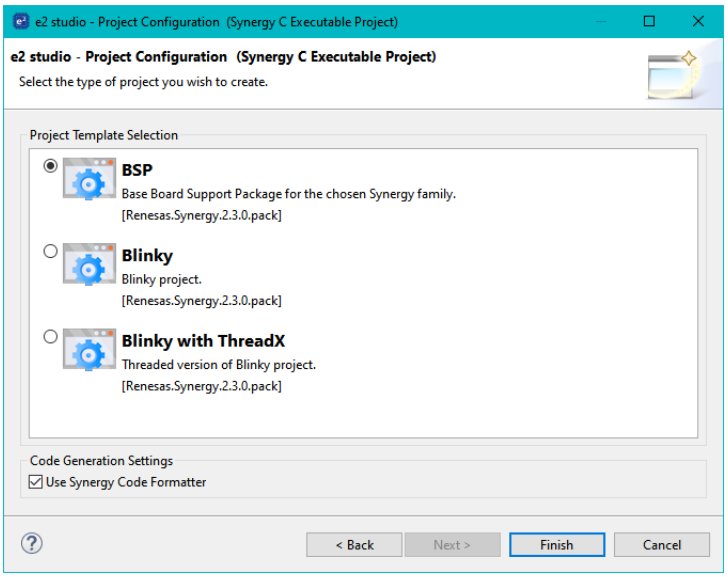


Figure 81: Project Configuration (Part 3)

4. Click **Finish**.

The ISDE creates the project and opens the **Project Explorer** and **Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

3.3.4 Configuring the Project with the ISDE

The e2 studio ISDE simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

The ISDE offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++** , **Synergy Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right of the ISDE.

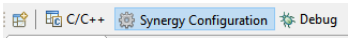


Figure 82: Selecting a Perspective

The **C/C++** perspective provides a layout selected for code editing. The **Synergy Configuration** perspective provides elements for configuring a Synergy project, and the **Debug** perspective provides a view suited for debugging.

1. In order to configure the project settings ensure the **Synergy Configuration** perspective is selected.
2. Ensure the **Project Configuration [WDT Application]** is open. It is already open if the Summary information is visible. To open the Project Configuration now or at any time make sure the **Synergy Configuration** perspective is selected and double-click on the **configuration.xml** file in the Project Explorer pane on the right side of the ISDE.

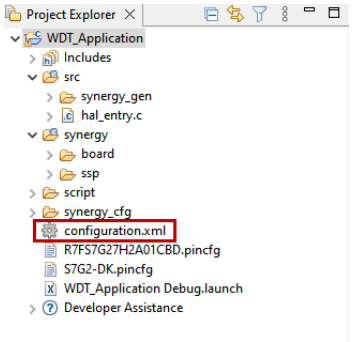


Figure 83: Synergy Project Configuration Settings

At the base of the Project Configuration view there are several tabs for configuring the project. A project may require changes to some or all of these tabs. The tabs are shown below.

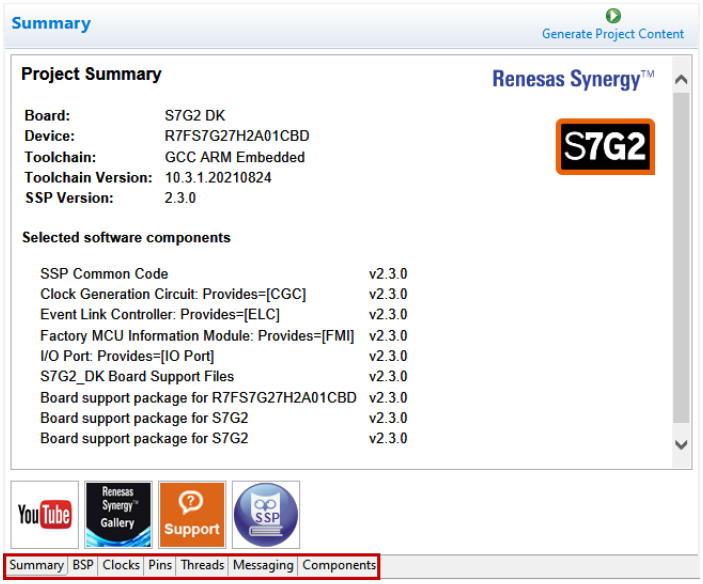


Figure 84: Project Configuration Tabs

3.3.4.1 BSP Tab

The BSP tab allows the Board Support Package (BSP) options to be modified from their defaults. For this particular WDT project no changes are required. However, if you want to use the WDT in auto-start mode, you can configure the settings of the OFS0 (Option Function Select Register 0) register in the BSP tab. See the Synergy Hardware User's Manual for details on the WDT autostart mode.

3.3.4.2 Clocks Tab

The clocks tab presents a graphical view of the clock tree of the device. Using the drop down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLCKB. The default output frequency for this clock is 60 MHz. Ensure this clock is outputting this value.

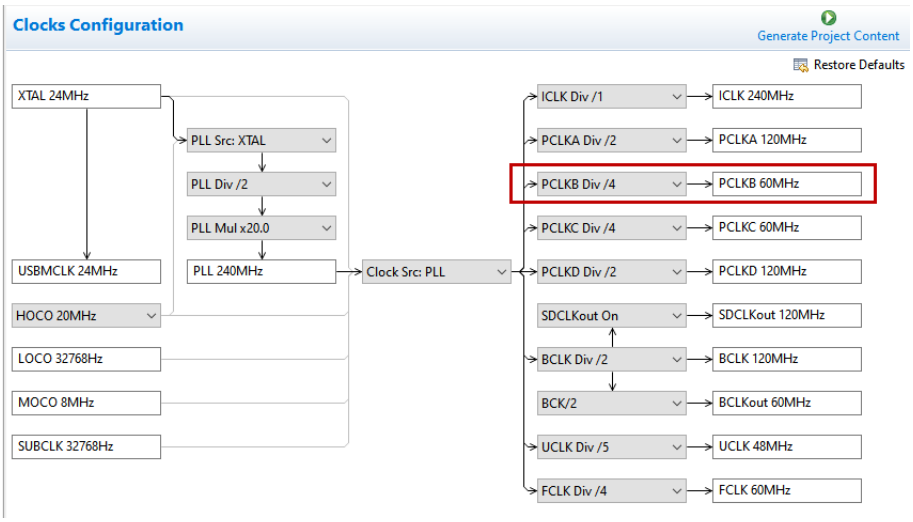


Figure 85: Clock Configuration

3.3.4.3 Pins Tab

The Pins tab provides a graphical tool for configuring the functionality of the pins of the device. For the WDT project no pin configuration is required. Although the project uses two LEDs connected to pins on the device, these pins are pre-configured as output GPIO pins by the BSP.

3.3.4.4 Threads Tab

You can add any driver to the project using the Threads tab. The HAL drivers for the Clock Generation Circuit, the Event Link Controller, and the IO port pins are added automatically by the ISDE when the project is configured. The WDT application uses no ThreadX Resources, so you only need to add the HAL WDT driver.

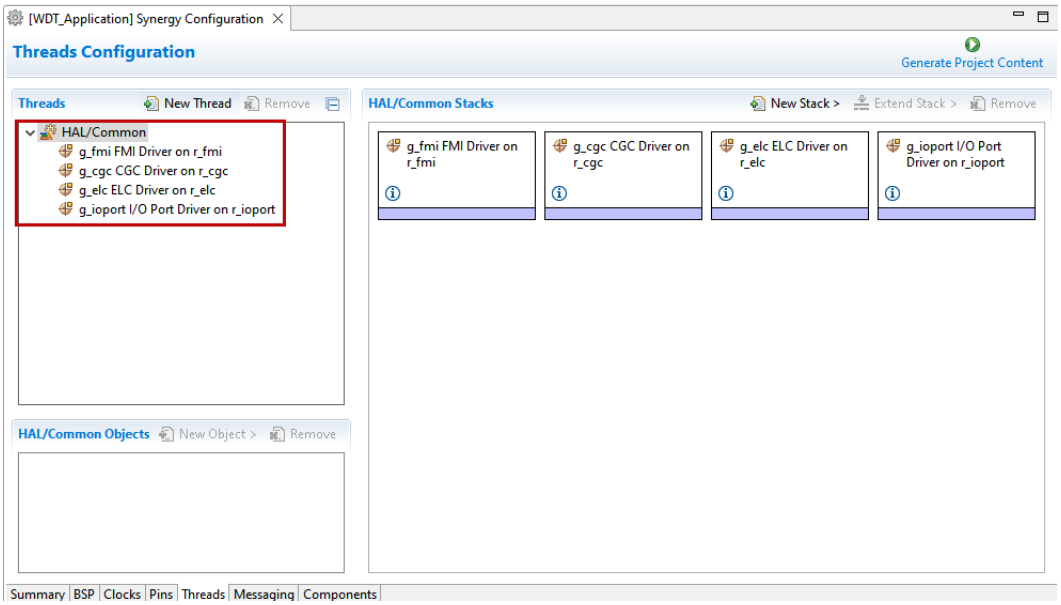


Figure 86: Threads Tab

The **HAL/Common Stacks** panel and is populated with the modules preselected by the ISDE.

1. Click on **New Stack** to find a pop-up window with the available HAL level drivers.
2. Select **Watchdog Driver on r_wdt**.

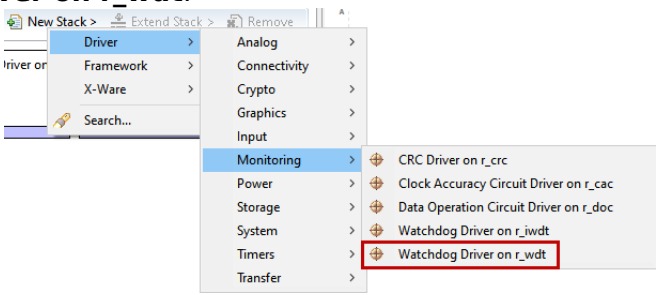


Figure 87: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Stacks** Panel and the **Property** Window shows all configuration options for the selected module. The **Property** tab for the WDT should be visible at the bottom left of the screen. If it is not visible check that the **Synergy Configuration** perspective is selected.

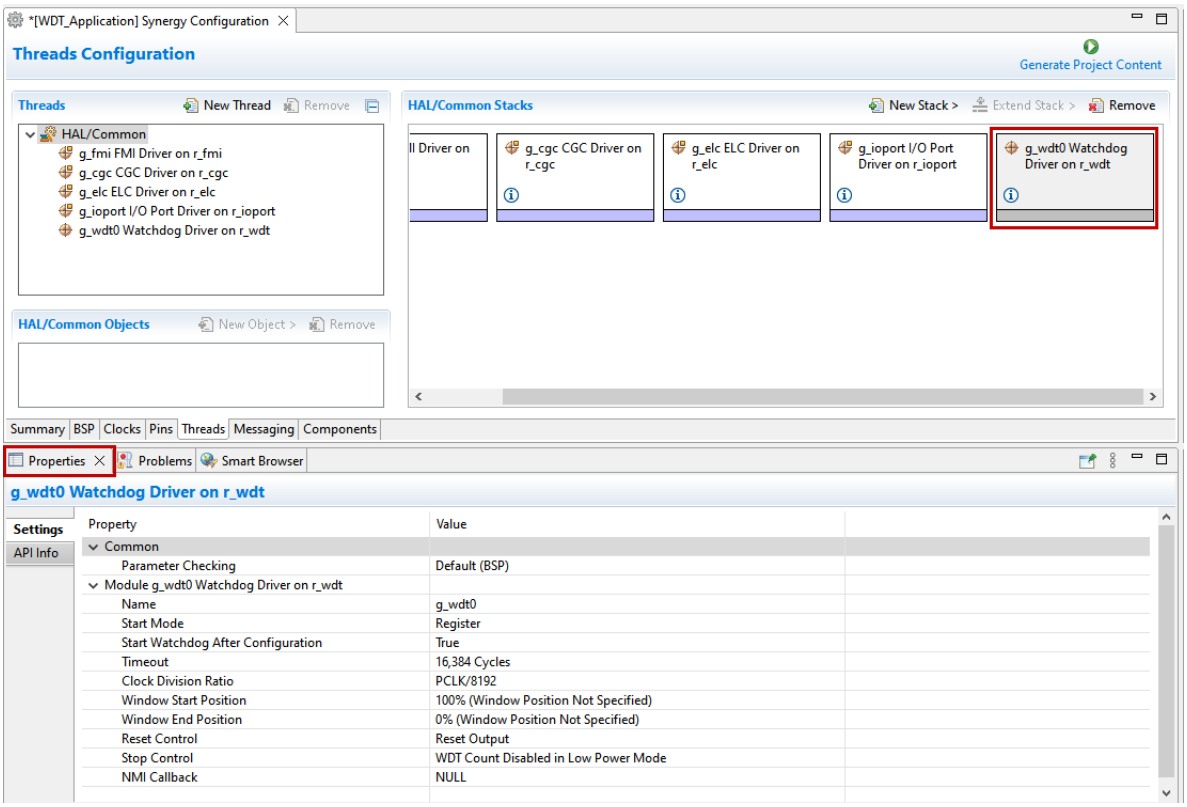


Figure 88: Module Properties

Change parameter **Start Watchdog After Configuration** from **True** to **False**. The other parameters can be left with their default values. Setting **Start Watchdog After Configuration** to **False** instructs the WDT driver (via its open API call) to configure the WDT but not to start it. It will be started later by refreshing it.

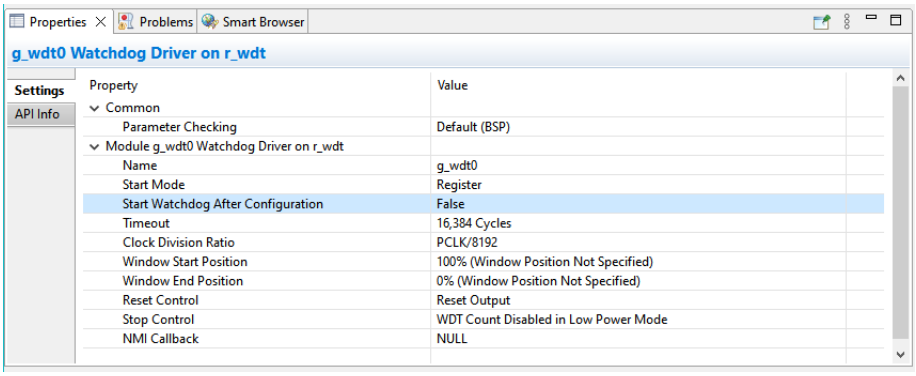


Figure 89: g_wdt Watchdog Driver on WDT Properties

With PCLKB running at 60 MHz the WDT will reset the device 2.23 seconds after the last refresh.

WDT clock = 60 MHz / 8192 = 7.32 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 = 2.23 seconds

Save the **Project Configuration** file and click the **Generate Project Content** button in the top right corner of the **Project Configuration** pane.

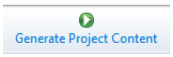


Figure 90: Generate Project Content Button

The ISDE generates the project files.

3.3.4.5 Components Tab

The components tab is included for reference to see which modules are included in the project. Modules are selected automatically in the Components view when after they are added in the Threads Tab.

For the WDT project ensure that the following modules are selected:

1. HAL_Drivers -> r_cgc
2. HAL_Drivers -> r_elc
3. HAL_Drivers -> r_ioport
4. HAL_Drivers -> r_wdt
5. HAL_Drivers -> r_fmi

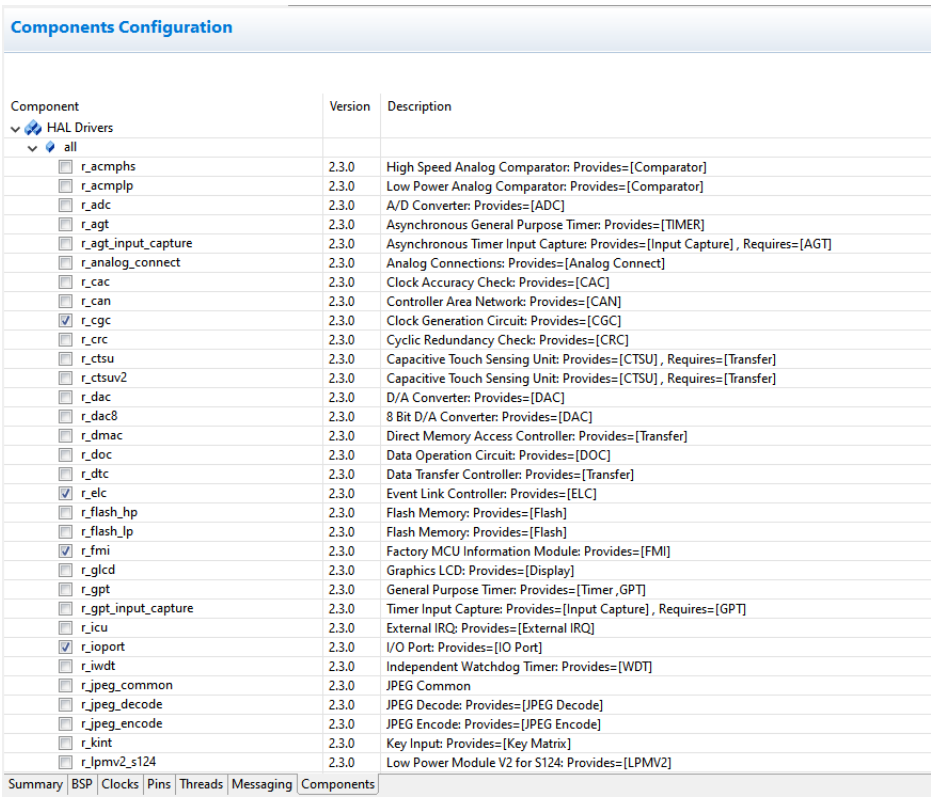


Figure 91: Component Selection

Note

The list of modules displayed in the Components tab depends on the installed SSP version.

3.3.5 WDT Generated Project Files

Pressing the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:
  - synergy/ssp/src/driver/
- r_wdt_api.h created in:
  - synergy/ssp/inc/driver/api
- r_wdt.h created in:
  - synergy/ssp/inc/driver/instances

The above files are the standard files for the WDT HAL module. They contain no specific project contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

Configuration information for the WDT HAL module in the WDT project is found in:

- synergy_cfg/ssp_cfg/driver/r_wdt_cfg.h

The above file's contents are based upon the **Common** settings in the **g_wdt Watchdog Driver on WDT Properties** pane.

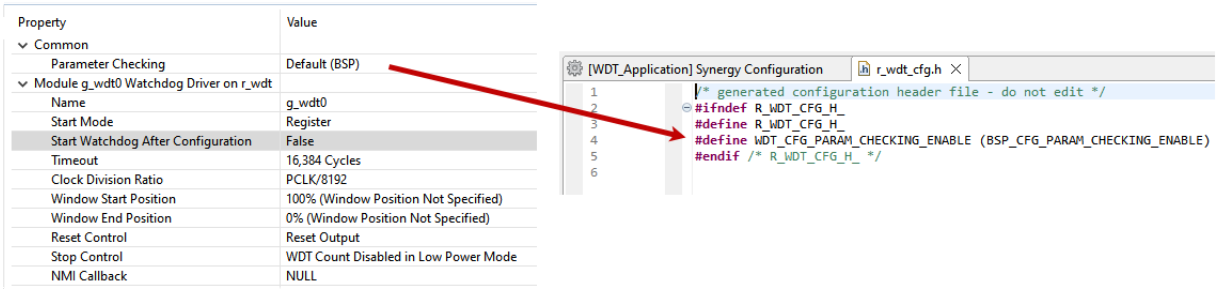


Figure 92: r_wdt_cfg.h Contents

Warning

Do not edit any of these files as they are recreated every time the Generate Project Content button is pressed and so any changes will be overwritten.

r_ioport folder is not created at ssp/src/driver as this module is required by the BSP and so already exists. It is included in the WDT project in order to include the correct header file in src/synergy_gen/hal_data.h – see later in this document for further details. For the same reason the other IOPORT header files – synergy/ssp/inc/api/r_ioport_api.h and synergy/ssp/inc/instances/r_ioport.h are not created as they already exist.

In addition to generating the HAL driver files for the WDT and IOPORT files the ISDE also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

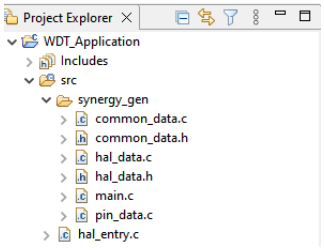


Figure 93: WDT Project Files

3.3.5.1 WDT hal_data.h

The contents of hal_data.h are shown below.

```
/* generated HAL header file - do not edit */
#ifndef HAL_DATA_H_
#define HAL_DATA_H_
#include <stdint.h>
#include "bsp_api.h"
#include "common_data.h"
#include "r_wdt.h"
#include "r_wdt_api.h"
```

```
#ifdef __cplusplus
extern "C"
{
#endif
extern const wdt_instance_t g_wdt0;
#ifndef NULL
void NULL(wdt_callback_args_t *p_args);
#endif
void hal_entry(void);
void g_hal_init(void);
#ifdef __cplusplus
} /* extern "C" */
#endif
#endif /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the ISDE generated project. In addition this file includes external references to the **g_wdt** instance structure which contains pointers to the configuration, control, api structures used for WDT HAL driver.

Warning

This file is regenerated each time Generate Project Content is pressed and must not be edited.

3.3.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */
#include "hal_data.h"
static wdt_instance_ctrl_t g_wdt0_ctrl;
static const wdt_cfg_t g_wdt0_cfg =
{ .start_mode = WDT_START_MODE_REGISTER,
  .autostart = false,
  .timeout = WDT_TIMEOUT_16384,
  .clock_division = WDT_CLOCK_DIVISION_8192,
  .window_start = WDT_WINDOW_START_100,
  .window_end = WDT_WINDOW_END_0,
  .reset_control = WDT_RESET_CONTROL_RESET,
```

```
  .stop_control = WDT_STOP_CONTROL_ENABLE,
  .p_callback = NULL, };
/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{ .p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt };
void g_hal_init(void)
{
    g_common_init ();
}
```

hal_data.c contains g_wdt_ctrl which is the control structure for this instance of the WDT HAL driver. This structure should not be initialised as this is done by the driver when it is opened.

The contents of **g_wdt_cfg** are populated in this file using the **g_wdt Watchdog Driver on WDT Properties** pane in the **ISDE Project Configuration HAL** tab. If the contents of this structure do not reflect the settings made in the ISDE, ensure the **Project Configuration** settings are saved in the ISDE before pressing the **Generate Project Content** button.

Warning
This file is regenerated each time Generate Project Content is pressed and so should not be edited.

3.3.5.3 WDT main.c

Contains main() called by the BSP start-up code. main() calls hal_entry() which contains user developed code (see next file). Here are the contents of main.c.

```
/* generated main source file - do not edit */
extern void hal_entry(void);
int main(void)
{
    hal_entry ();
 return 0;
}
```

Warning
This file is regenerated each time Generate Project Content is pressed and so should not be edited.

3.3.5.4 WDT hal_entry.c

This file contains the function hal_entry() called from main(). User developed code should be placed in this file and function.

For the WDT project edit the contents of this file to contain the code below. This code implements the flowchart in overview section of this document.

```
/* HAL-only entry function */
#include "hal_data.h"
#define RED_LED_NO_OF_FLASHES 30
#define RED_LED_PIN IOPORT_PORT_06_PIN_01 ? In case of SK-S7G2
#define GREEN_LED_PIN IOPORT_PORT_06_PIN_00 ? In case of SK-S7G2
#define RED_LED_DELAY_COUNT 1500000
#define GRN_LED_DELAY_COUNT 1200000
volatile uint32_t delay_counter;
volatile uint16_t loop_counter;
void hal_entry(void)
{
 /* TODO: add your own code here */
 /* Open the WDT */
    g_wdt0.p_api->open(g_wdt0.p_ctrl, (wdt_cfg_t *const)g_wdt0.p_cfg);
 /* Start the WDT by refreshing it */
    g_wdt0.p_api->refresh(g_wdt0.p_ctrl);
 /* Flash the red LED and tickle the WDT for a few seconds */
 for(loop_counter=0; loop_counter<RED_LED_NO_OF_FLASHES; loop_counter++)
    {
 /* Turn red LED on */
       g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_HIGH);
 /* Delay */
 for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++);
 /* Refresh WDT */
      g_wdt0.p_api->refresh(g_wdt0.p_ctrl);
 /* Turn red off */
       g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_LOW);
 /* Delay */
 for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++);
 /* Refresh WDT */
```

```
      g_wdt0.p_api->refresh(g_wdt0.p_ctrl);
    }
 /* Flash green LED but STOP tickling the WDT. WDT should reset the
  device */
 while(1)
    {
 /* Turn green LED on */
       g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_HIGH);
 /* Delay */
 for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);
 /* Turn green off */
       g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_LOW);
 /* Delay */
 for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);
    }
}
```

The WDT HAL driver is called through the interface **g_wdt_on_wdt** defined in **r_wdt.h**. The WDT HAL driver is opened through the open API call using the instance defined in **r_wdt_api.h**:

```
g_wdt0.p_api->open(g_wdt0.p_ctrl, (wdt_cfg_t *const)g_wdt0.p_cfg);
```

The first passed parameter is the pointer to the control structure **g_wdt_ctrl** instantiated in hal_data.c. The second parameter is the pointer to the configuration data **g_wdt_cfg** instantiated in the same hal_data.c file.

The WDT is started and refreshed through the API call:

```
g_wdt0.p_api->refresh(g_wdt0.p_ctrl);
```

Again the first (and only in this case) parameter passed to this API is the pointer to the control structure of this instance of the driver.

3.3.6 Building and Testing the Project

Build the project in the ISDE **Build > Build Project**. The project should build without errors.

To debug the project

1. Connect the JLink debugger between the target board and host PC. Apply power to the board.
2. In the **Project Explorer** pane on the right side of the ISDE right-click on the WDT project **WDT_Application** and select **Debug As > Debug Configurations**.
3. Under **Renesas GDB Hardware Debugging** select **WDT_Application Debug** as shown below.

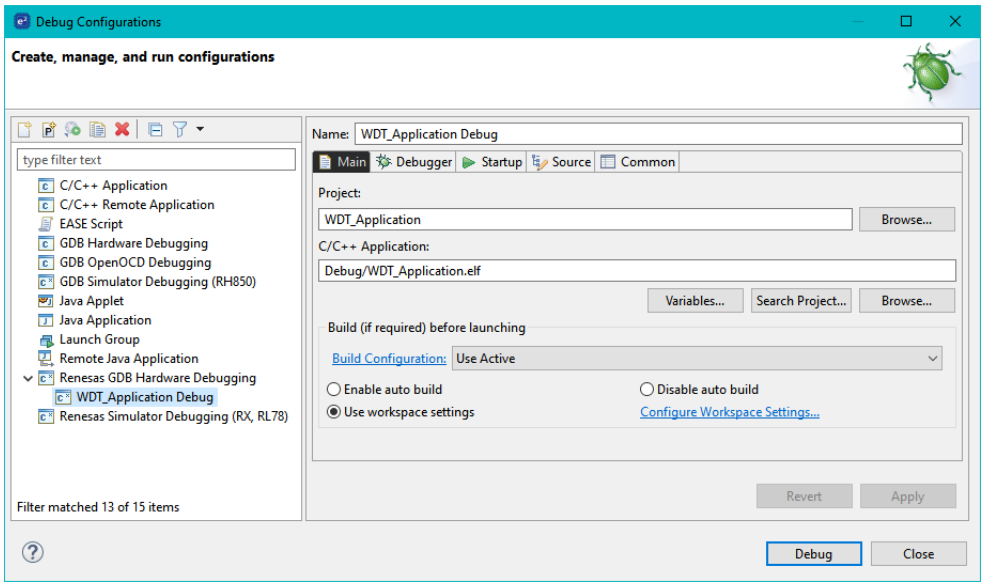


Figure 94: Debug Configuration

4. Press the **Debug** button. Switch (Yes) to the Debug perspective if asked.
5. The code should run to the Reset_Handler() function.
6. Resume execution via **Run > Resume**. Execution will stop in main() at the call to **hal_entry()**.
7. Resume execution again.

The red LED should start flashing. After 30 flashes the green LED will start flashing and the red LED will stop flashing.

While the green LED is flashing the WDT will underflow and reset the device resulting in the red LED to flash again as the sequence repeats. However, this sequence does not occur when using the debugger because the WDT does not run when connected to the debugger.

1. Stop the debugger in the ISDE via **Run > Terminate**.
2. Press the reset button on the target board. The LEDs begin flashing.

3.4 IAR Embedded Workbench for Renesas

3.4.1 Using IAR Embedded Workbench for Synergy

This section describes how to use the IAR Embedded Workbench for Renesas Synergy (IAR EW for Synergy) in combination with the Renesas Synergy Standalone Configurator (SSC) to develop

applications with the Renesas Synergy Software Package (SSP). The architecture of the SSP directly determines how you use the IAR EW for Synergy and SSC to develop a Synergy application. See the following documents for details on the SSP architecture included in this manual:

- SSP Architecture
- BSP Architecture

3.4.2 What is IAR EW for Synergy?

IAR Embedded Workbench is now completely integrated with the Renesas Synergy Platform. The new product IAR EW for Synergy provides add-on functionality to simplify and accelerate software development, and provide the best performance and smallest code size.

Just like e^2 studio, IAR EW for Synergy offers secure source-level visibility into the Synergy Software Package (SSP) as well as secure source-level debugging. The developer can see protected source code but not modify or save it. Once the application code is developed, IAR EW for Synergy includes IAR C-STAT® and C-RUN® analyzers, tools which help and guide to improve application code quality.

3.4.3 IAR EW Key Features

- Integrated development environment with project management tools and editor
- Highly optimizing C and C++ compiler and Linker for Renesas Synergy devices
- Integration support for Renesas Synergy Standalone Configurator (SSC)
- C-STAT and C-RUN code analysis tools included
- Extensive HW target system support
- Power debugging to visualize power consumption in correlation with source code
- C-SPY® Debugger with JTAG/SWD support and support for RTOS-aware debugging on hardware
- Support for ETM Trace
- Comprehensive user and reference guides and context-sensitive help function
- Compliant with ARM® Embedded Application Binary Interface (EABI) and ARM Cortex® Microcontroller Software Interface Standard (CMSIS)

For detailed instructions on how to download and install IAR EW for Synergy, see the IAR EW for Synergy Release Notes on the Synergy Gallery.

3.4.4 What is Synergy Standalone Configurator (SSC)?

The Synergy Standalone Configurator (SSC) is an Eclipse Rich Client Platform (RCP) application containing the Synergy Project Generator and the Synergy Project Editor as implemented in the Renesas e^2 studio ISDE. SSC includes configurators like the Clock Configurator, Pin Configurator, RTOS Configurator, SSP Module Selector/Configurator, and Interrupt Control Unit (ICU) Configurator for use with 3rd party IDEs such as IAR EW for Synergy.

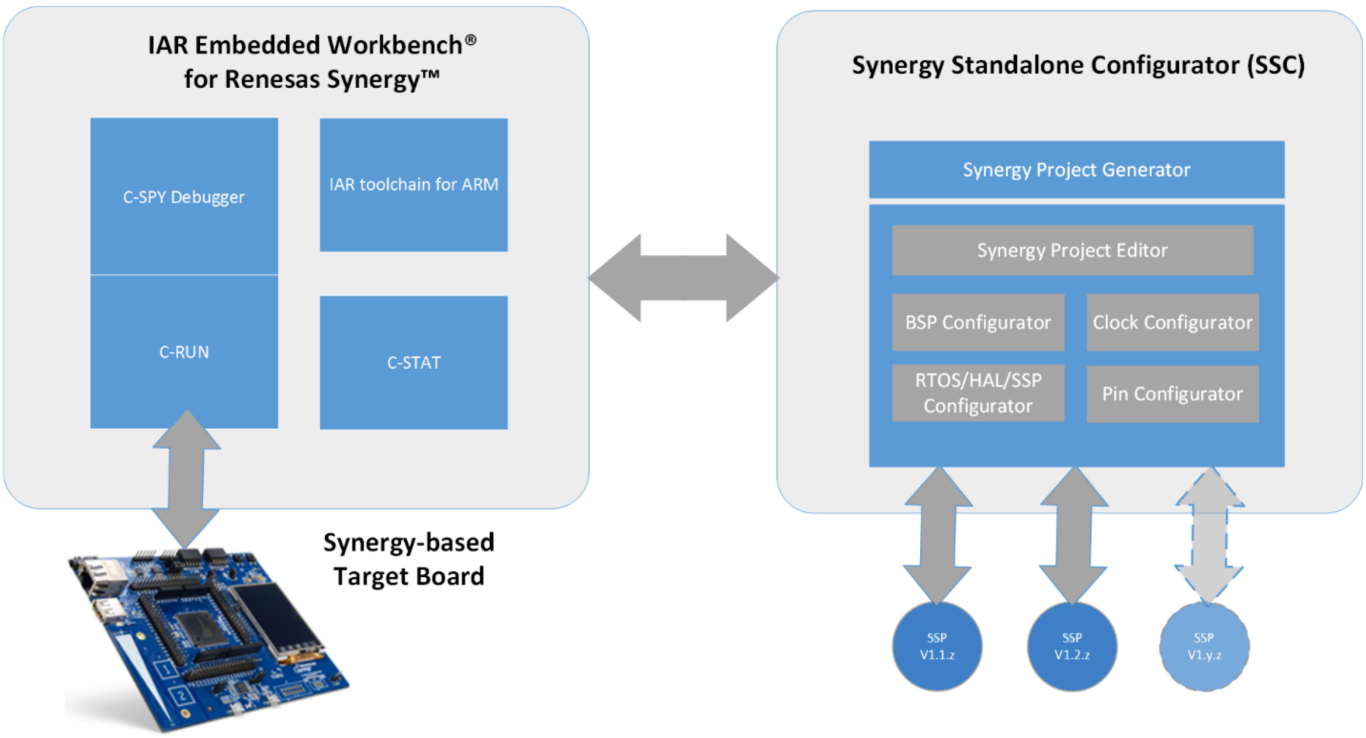


Figure 95: IAR EW for Synergy and SSC Functional Block Diagram

Since the functionality of the SSC is identical to the Synergy Project Generator and the Synergy Project Editor as implemented in the Renesas e^2 studio ISDE, refer to e2 studio ISDE User Guide for information on how to use it.

For detailed instructions on how to download and install the SSC and the SSP to use with IAR EW for Synergy, see the SSC Release Notes and the SSP Release notes on the Synergy Gallery.

3.4.5 Installing the Tools

To install the tools, follow the steps below:

1. Download and install the Renesas Synergy Standalone Configurator (SSC) from the Renesas Synergy Gallery. You can find it under Development Tools. The default installation directory is C:\Renesas\Synergy\SSC_<SSCversion>.
2. Download and install the Renesas Synergy Software Package (SSP) from the Renesas Synergy Gallery. During the installation you will be prompted to specify an installation directory for the SSP. Point the SSP installer to the directory where you just installed the SSC (for example C:\Renesas\Synergy\SSC_<SSCversion>).
3. Download and install IAR Embedded Workbench for Renesas Synergy from the Renesas Synergy Gallery. To install IAR Embedded Workbench:
    a. In your web browser, specify the URL https://synergygallery.renesas.com and download IAR Embedded Workbench for Renesas Synergy from the Renesas Synergy Gallery. You will also find information about how to obtain a license and get a license number.
    b. Execute the installer that is included in the downloaded file.
    c. Specify the license number when prompted for in the IAR License Manager.

Note

The IAR EW for Synergy license entitles you to use this specific edition of IAR Embedded Workbench, but not the Synergy Standalone Configurator for which separate licenses are required.

3.4.6 Creating a Renesas Synergy Project using IAR EW for Synergy and SSC

To create a Synergy Project using IAR EW for Synergy and SSC, follow the steps below:

1. In the IAR Embedded Workbench IDE, choose **Project>Create New Project**.

2. In the **Create New Project** dialog box, select **Renesas Synergy Project** and click **OK**.

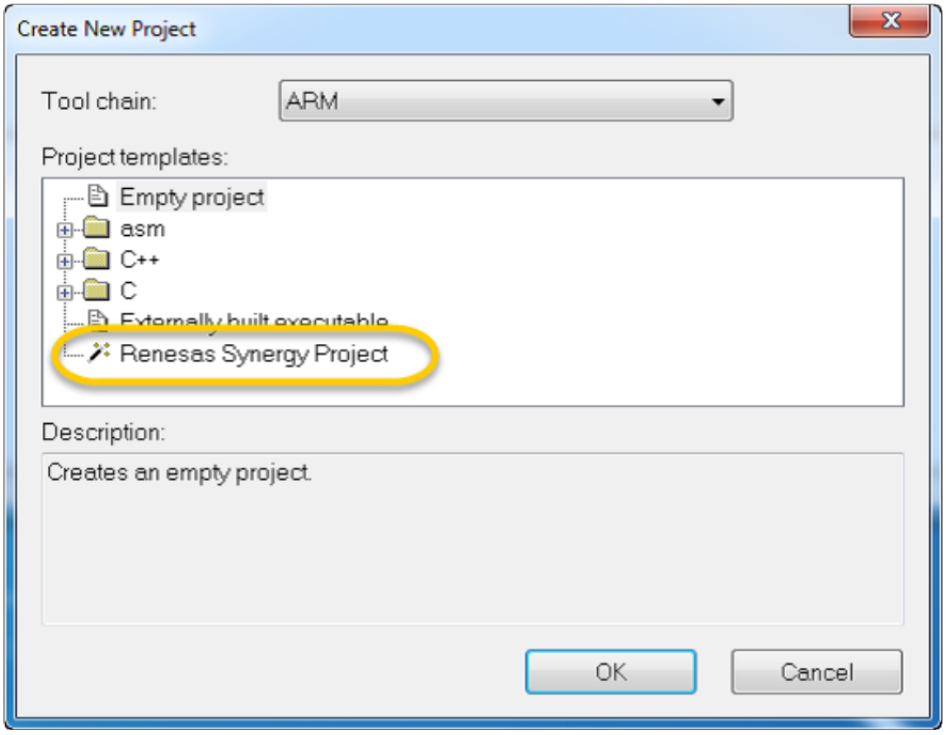


Figure 96: Creating a New Synergy Project using IAR EW for Synergy

3. In the **Save As** dialog box that appears, choose a suitable destination directory for your workspace (the container that holds your project), for example MyWorkspace, and click **Save**.

Note

Do not save your workspace in the root directory of your operating system (C:).

4. In the **Renesas Synergy Setting** dialog box that appears, specify the location of your installed Synergy Standalone Configurator (SSC), which by default is installed in C:\Renesas\Synergy\SSC_<SSCversion>.

Note

*You do not need to specify a license file. You can click **OK** with the **License file** field empty. An SSP license file is not required and the source files are not encrypted for SSP v2.0.0 and later.*

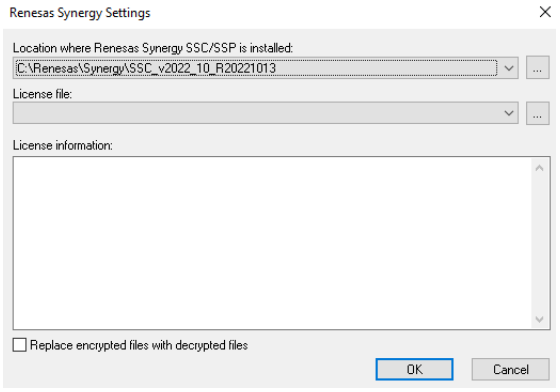


Figure 97: Renesas Synergy Settings in IAR EW for Synergy

5. Click **OK**.

6. In the **Save As** dialog box that appears, specify the name of your project, for example MyProject.

Note

Do not save your project in the root directory of your operating system (C:).

7. The IAR Embedded Workbench IDE now connects with the Renesas Synergy Standalone Configurator (SSC). Specify the board support you require:

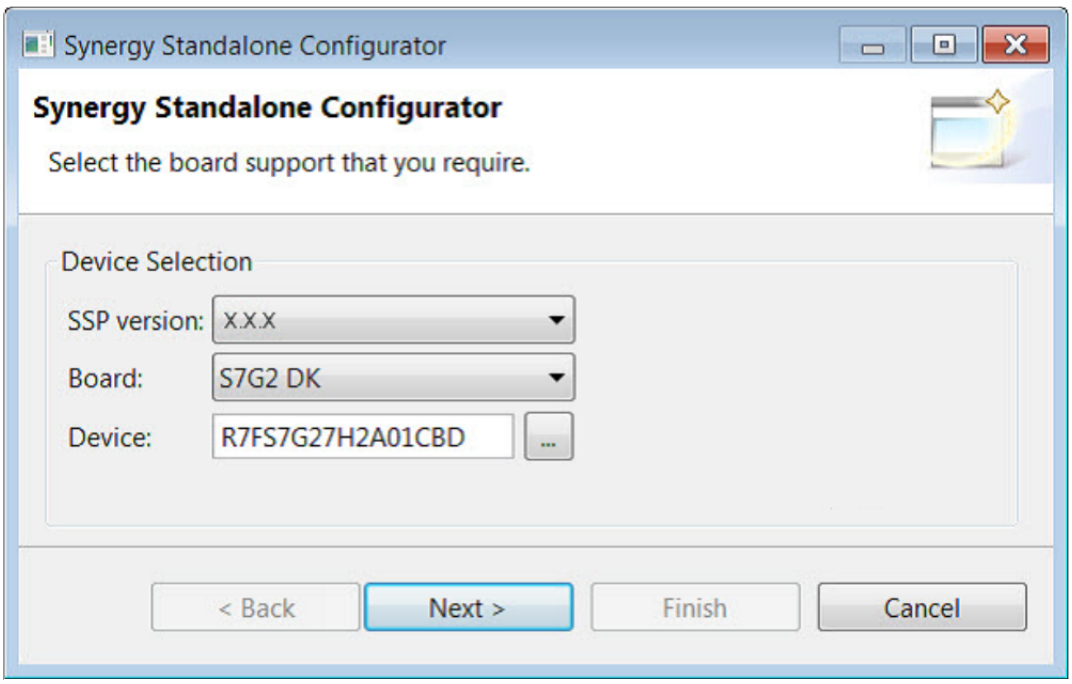


Figure 98: SSC selection Dialog for SSP, Board, and MCU to be used

Note

The SSP versions available in the drop-down list correspond to the versions you have previously

installed on your computer.

Click **Next**.

8. The Synergy Software Packages come with several example projects, which include source code files, header files, and linker configuration files, adapted for your device. Select the example packages that you want to add to your project:

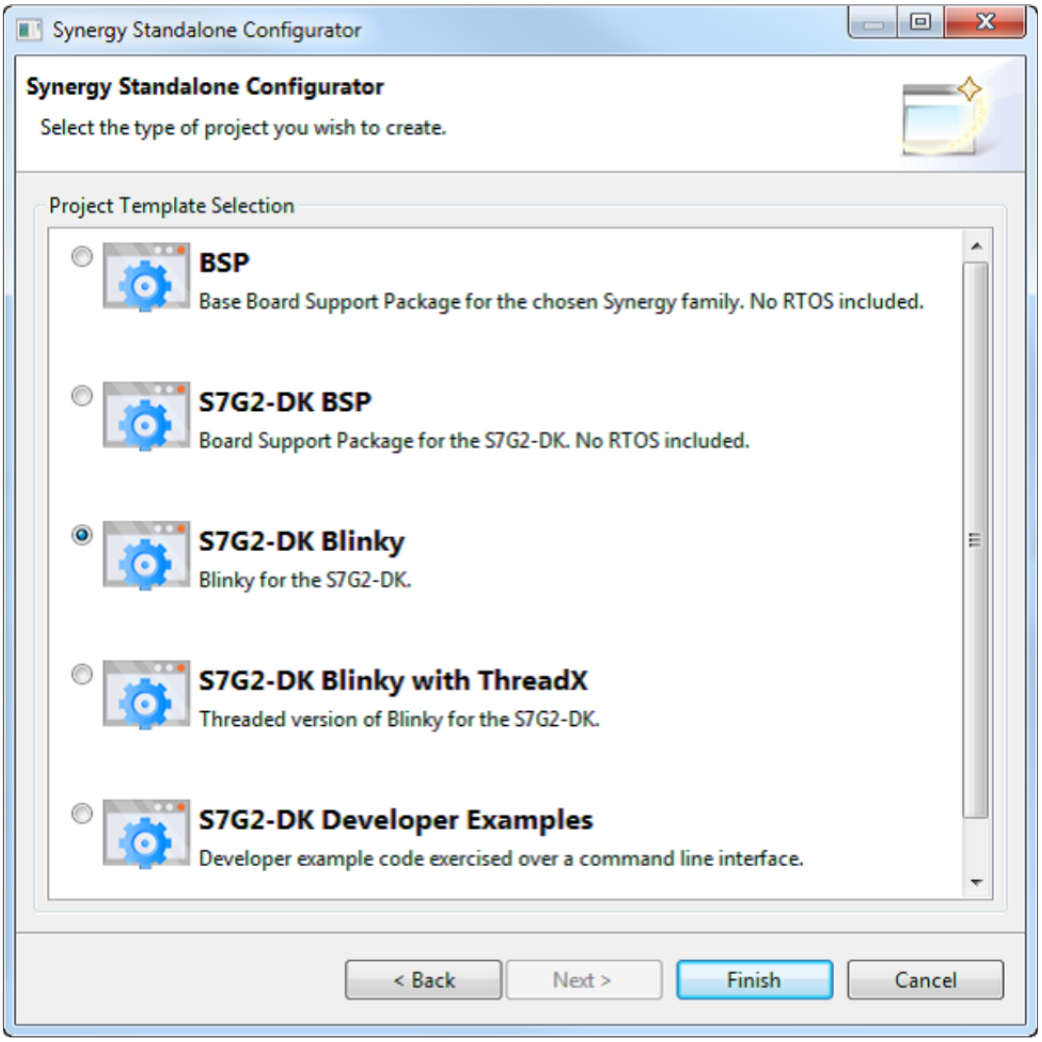


Figure 99: Project Template Selection Dialog

Click **Finish**.

9. In the Synergy Project Editor that opens up, you can now configure MCU pin function assignments, clock and peripheral settings, and interrupt source assignments. When finished configuring, click the **Generate Project Content** button. The source code is now generated.

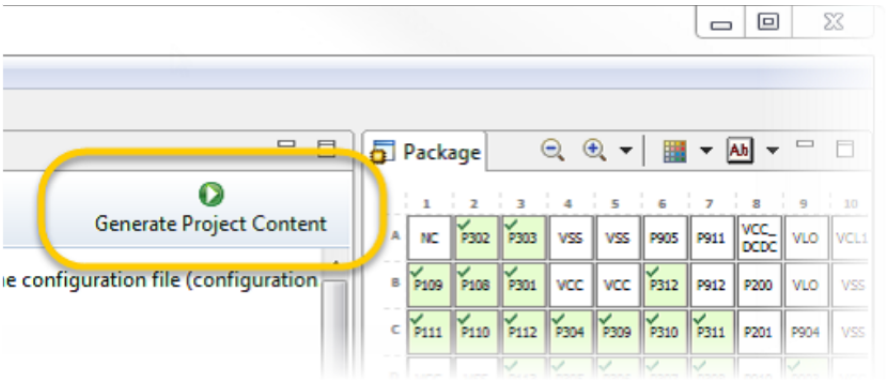


Figure 100: Generate Project Content Button

Note

You can always add or change the configuration of your Synergy project later on.

10. After a couple of seconds, your Renesas Synergy project is displayed in the IAR EW Workspace window:

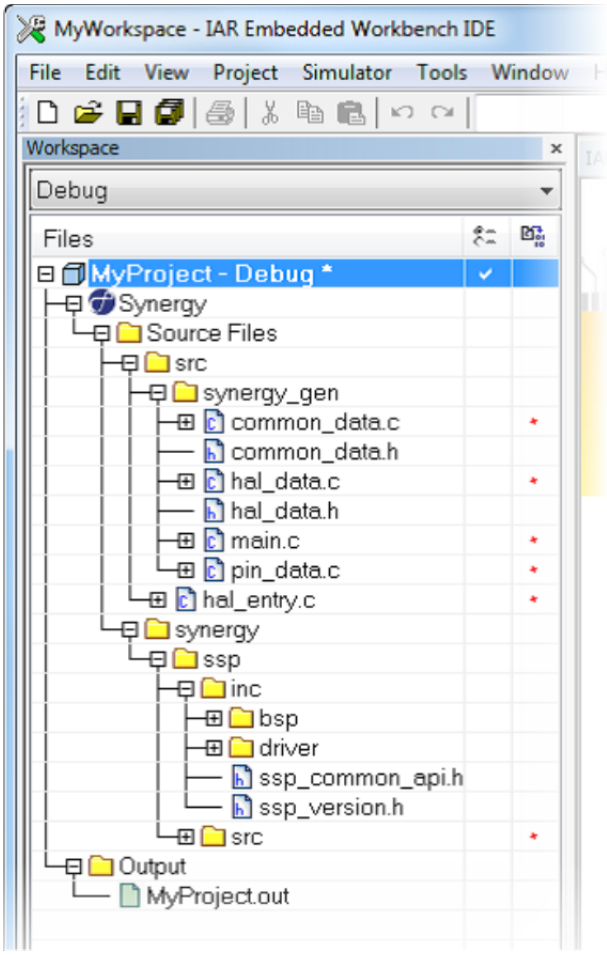


Figure 101: IAR EW for Synergy Workspace and Project Window

11. If you close the SSC, you can re-open it again by clicking the Synergy Configuration button

in the toolbar, or by selecting **Renesas Synergy > Configurator** from the menu.

12. Whenever you switch back to the SSC to change configuration settings, click the **Generate Project Content** button when finished. The affected source code files are now re-generated.

13. You can now continue building and debugging according to the standard routines in the IAR Embedded Workbench IDE, see the IAR Embedded Workbench® IDE User Guide on the IAR web site.

    As the SSC works just like the Synergy Project Editor in e^2 studio, refer to e2 studio ISDE User Guide for more details on how to use it.

    Note that the Synergy Project is by default configured for the J-Link debugging probe.

    If you have another debugging probe, for example I-jet or I-jet Trace, choose **Project>Options>Debugger** in the IDE and select I-jet/JTAGjet from the Driver drop-down list.

Chapter 4 Module Overviews

You can find a list of Module Overviews on the following pages:

- Framework Layer
- HAL Layer
- Azure RTOS Modules

The Module Overviews for each SSP module have been significantly improved since the last release. The new Module Overviews should provide all the information necessary for a developer to evaluate a specific modules fitness for use in a target application and significantly help with the development process. The intent of these notes is to provide all the information needed to begin develop0ment with the target module in one easy to find location.

Each Module Overview includes the following sections:

1. A short introduction to the module includes a short description, a block diagram of key module components, and a list of features
2. An API table lists all the available APIs, and example use of the API and a short description of the APIs function. A list of some of the key Status Return values is provided to help determine the result of the API call.
3. A functional overview describes key module operations and includes a list of important module limitations.
4. A step-by-step description of how to include the module in an application using the ISDE threads tab and stack selection process.
5. A set of tables showing the configuration parameters for the module and key lower level modules is provided so the developer can easily see the modules key capabilities. Note that these configurable properties vary by MCU series and by SSP Release. Treat the tables in these notes as illustrations and refer to the actual parameters available within the ISDE for your target MCU and for your chosen SSP release. Example pin and clock configuration and selection information is also provided to help guide development.
6. A simple implementation using the target module is provided and shows the steps used in a typical application, the associated flow diagram, and the API use at each step. This helps describe how APIs are commonly used and will give the developer a head start with their implementation.

Module Guide Application Notes

These six sections are also found in the Module Guide Application Notes for a specific module. In the Application Note additional sections provide detailed descriptions of the associated application project that demonstrates the module working in an actual design. Development can be dramatically simplified when the application project is used as a starting point or reference for a new design. Module guide application notes can be found with the following search: https://www.renesas.com/us/en/support/document-search?doc_file_all_types%5BApplication+Note%5D=Application+Note&doc_file_all_types%5BSample+Code+-+FIT+Module%5D=Sample+Code+-+FIT+Module&doc_file_all_types%5BSample+Code%5D=Sample+Code&doc_category_tier_1=467666&doc_category_tier_2=469306&doc_part_numbers=&keywords=&sort_order=DESC&sort_by=field_document_revision_date#documentation-tools-results. More module guide application notes are being added all the time so check back frequently to find when new ones have been released.

Using the Module Guide Module Overviews

The Module Overviews provide sufficient details to begin development, but there will be cases when additional information is useful in implementing a design. The SSP User Manual provides a wealth of information on the details of API implementation, structures, enumerations and more. Simply jump to the API reference section and find your module of interest to find any additional information you might need.

HAL modules have chapters that cover the above topics as well. It is highly recommended that you spend some time looking at the reference material available in ALL the reference chapters so you know where to look when an API implementation question, not answered in the module guide usage note or associated module guide application note, comes up.

4.1 Framework Layer

Some SSP framework modules are not included in the following list of Module Overviews. This is because some modules, although they can be selected and added to a thread, are only used as lower level modules and are not expected to be used by a developer separately. These modules are included in the Module Overview for the higher level module however. So if additional information is desired, just refer to the associated higher level Module Overview. The following list shows how to find these 'missing' modules:

Module	Included In
D/AVE 2D Port on sf_tes_2d_drw	GUIX Port on sf_el_gx under the Azure RTOS Modules section as GUIX Port
D/AVE 2D Driver on dave2d	GUIX Port on sf_el_gx under the Azure RTOS Modules section as GUIX Port

ADC Periodic Framework

Audio Playback Framework

Audio Playback Hardware Framework Shared on sf_audio_playback_hw_dac

Audio Playback Hardware Framework Shared on sf_audio_playback_hw_i2s

Audio Record ADC Framework

Audio Record I2S Framework

Block Media Framework on sf_block_media_lx_nor

Block Media Framework on sf_block_media_qspi

Block Media Framework on sf_block_media_ram

Block Media Framework on sf_block_media_sdmmc

BLE Framework

Cellular Framework

Telnet Communications Framework on sf_comms_telnet

Communications Framework on sf_el_ux_comms_v2

Console Framework

Crypto Framework

Capacitive Touch v2 Framework

External IRQ Framework

I2C Framework

JPEG Decode Framework

Memory Framework on sf_memory_qspi_nor

Messaging Framework

Power Profiles V2 Framework

SPI Framework

Thread Monitor Framework

Touch Panel V2 Framework

UART Communications Framework

Wi-Fi Framework

Wi-Fi QCA4010 Framework

4.1.1 ADC Periodic Framework

4.1.1.1 ADC Periodic Framework Module Introduction

The ADC Periodic Framework provides a high-level API for signal processing applications. The module configures the ADC/SDADC to sample any of the available channels (using the single-scan mode) at a configurable rate and buffers the data for a configurable number of sampling iterations before notifying the application. The ADC Periodic Framework uses the ADC/SDADC, GPT or AGT and DTC peripherals on a Renesas Synergy™ Microcontroller. A user-defined callback can be created to

process the data each time a new sample is available.

ADC Periodic Framework Module Features

- 24-bit Sigma-Delta A/D Converter (S1JA only).
- 16-bit A/D Converter (S1JA)
- 14-bit A/D Converter (S3A7, S3A6, S3A3, S124, S128)
- 12-bit A/D Converter (S7G2, S5D9, S5D5)
- Multiple Operation Modes
  - Single Scan
  - Group Scan
  - Continuous Scan
- Multiple Channels
  - 1 channel (S1JA)
  - 13 channels (unit 0), 12 channels (unit 1) (S7G2 and S5D9)
  - 13 channels (unit 0), 9 channels (unit 1) (S5D5)
  - 18 channels (S124)
  - 21 channels (S128)
  - 25 channels (S3A6)
  - 28 channels (S3A7)
  - Temperature sensor channel
  - Voltage sensor channel

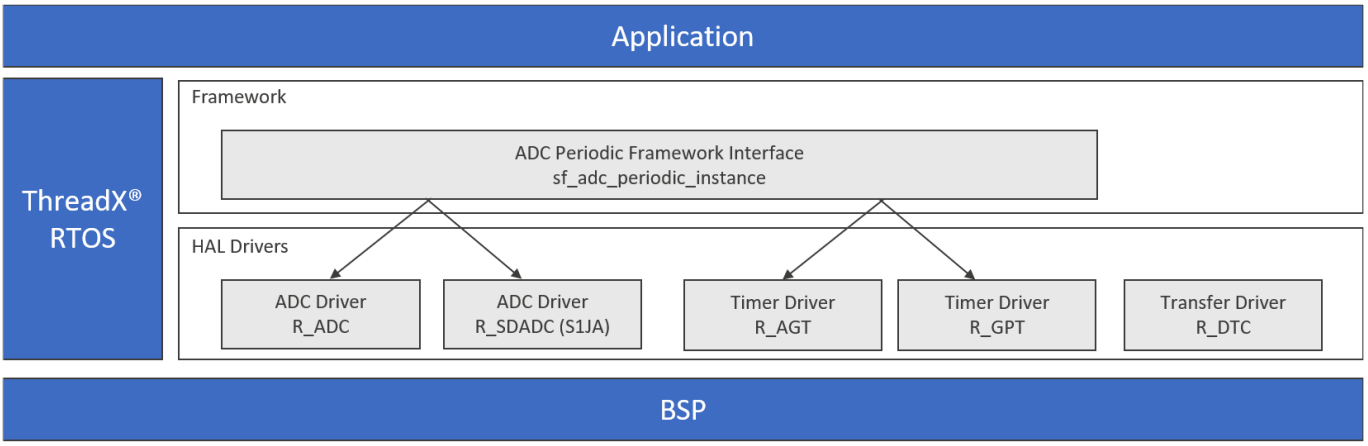


Figure 102: ADC Periodic Framework Module Block Diagram

4.1.1.2 ADC Periodic Framework Module APIs Overview

The ADC Periodic Framework defines APIs for opening, closing, starting and stopping the ADC scans. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ADC Periodic Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_adc_periodic.p_api->open(g_sf_adc_periodic.p_ctrl, g_sf_adc_periodic.p_cfg); Acquires mutex, then initializes module at the HAL layer.

start	g_sf_adc_periodic.p_api->start(g_sf_adc_periodic.p_ctrl); Starts the scan.
stop	g_sf_adc_periodic.p_api->stop(g_sf_adc_periodic.p_ctrl); Stops the hardware trigger (timer) from triggering any more ADC scans.
close	g_sf_adc_periodic.p_api->close(g_sf_adc_periodic.p_ctrl); Releases channel mutex and closes channel at HAL layer.
versionGet	g_sf_adc_periodic.p_api->versionGet(&version); Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.
SSP_ERR_NOT_OPEN	Driver control block not valid. CallSF_ADC_PERIODIC_Open to configure.
SSP_ERR_ASSERTION	Version get error- p_version was NULL.
SSP_ERR_INTERNAL	An internal ThreadX® error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.1.3 ADC Periodic Framework Module Operational Overview

The ADC Periodic Framework module samples and buffers ADC data. The Framework notifies the application once the configured number of samples are buffered. The ADC Periodic Framework works as follows:

- After initial configuration and after the scan process is started, the framework uses a hardware timer to trigger an ADC scan in one-shot mode. Each scan can consist of one or more channels. When each scan is completed, the ADC interrupt is intercepted by the DTC, which moves the result of the scan into the user buffer.
- Each scan is defined as a sampling iteration, and the number of samples generated for each scan is equal to the number of channels. If the channels are sequential, for example, channels 1, 2, 3, 4, the data is captured in order. If the channels are not in sequence, for

example, channels 1, 3, 4, 5, then the samples generated by each scan also include data from the unused channels in between. Thus, in the second example, five samples are stored to the user buffer each time.
- The user specifies the total number of sample iterations that need to occur before being notified. When the specified number of sampling iterations have occurred and the data for each iteration has been stored into the user buffer, the user is notified via a callback with an index for the valid data in the buffer and an event indicating that sampling for the specified number of iterations is complete.

Unless the user stops the scan process, the scan continues to be triggered by the timer (using AGT or GPT) and data will be written into the user buffer, which is treated by the Framework as a circular buffer. The name and length of the buffer are specified via the ISDE configurator.

ADC Periodic Framework Module Important Operational Notes and Limitations

ADC Periodic Framework Module Operational Notes

1. At least one channel must be chosen while configuring the ADC/SDADC HAL driver to avoid an API return error.
2. When configuring the scan rate for the ADC Framework (the GPT or AGT timer period), make sure that the period is long enough to accommodate scanning of all selected channels (about 2 microseconds for each channel conversion on a Synergy S7G2 device).
3. The ADC Periodic Framework stores data for all the channels from each scan into the user specified buffer. When the specified number of sample iterations are completed, the user is notified. If five channels are selected (channels 1,2,3,4,5) and the sample count is set to 3, the user will be notified when 5 x 3 = 15 samples are available. The samples are ordered as follows:

Figure 103: ADC Periodic Framework Module Sample Order

When selecting the data buffer length in the ADC Periodic Framework configuration, make sure that the buffer length is at least twice the length of the number of samples that will be generated (15 x 2 = 30 in this example). This is because once the user application is notified that the data is available, the Framework will keep buffering in new data at the sample rate. Since the buffer is treated as a circular buffer, you can inadvertently overwrite the data. If the size is not larger than the number of samples generated, the data is overwritten before the application can use it.

The application callback has an index into the appropriate location in the buffer where valid data is present.

ADC Periodic Framework Module Limitations

- The ADC Periodic framework does not currently support the following features:
  - The use of Group Scan mode
  - The use of DMA
- When configuring the ADC channels to be used with this framework, the temperature or voltage sensors must not be selected if any of the other available channels are also selected. It is possible to use only the temperature sensor, only the voltage sensor, or any number of the regular ADC channels.
- ADC Periodic framework does not support DTC transfer when lower lever driver is SDADC.
- When using ADC Periodic framework with lower level SDADC of 24-bit, user should not

access output data through "p_args" in callback function. User should access output data only through user defined buffer.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.1.4 Including the ADC Periodic Framework Module in an Application

This section describes how to include the ADC Periodic Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ADC Periodic Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the ADC Periodic Framework is g_adc_periodic0. This name can be changed in the associated Properties window.)

ADC Periodic Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_adc_periodic0ADC Periodic Framework on sf_adc_periodic	Threads	**New Stack**> **Driver**> **Analog**> **ADC Periodic Framework on sf_adc_periodic**

When the ADC Periodic Framework on sf_adc_periodic is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

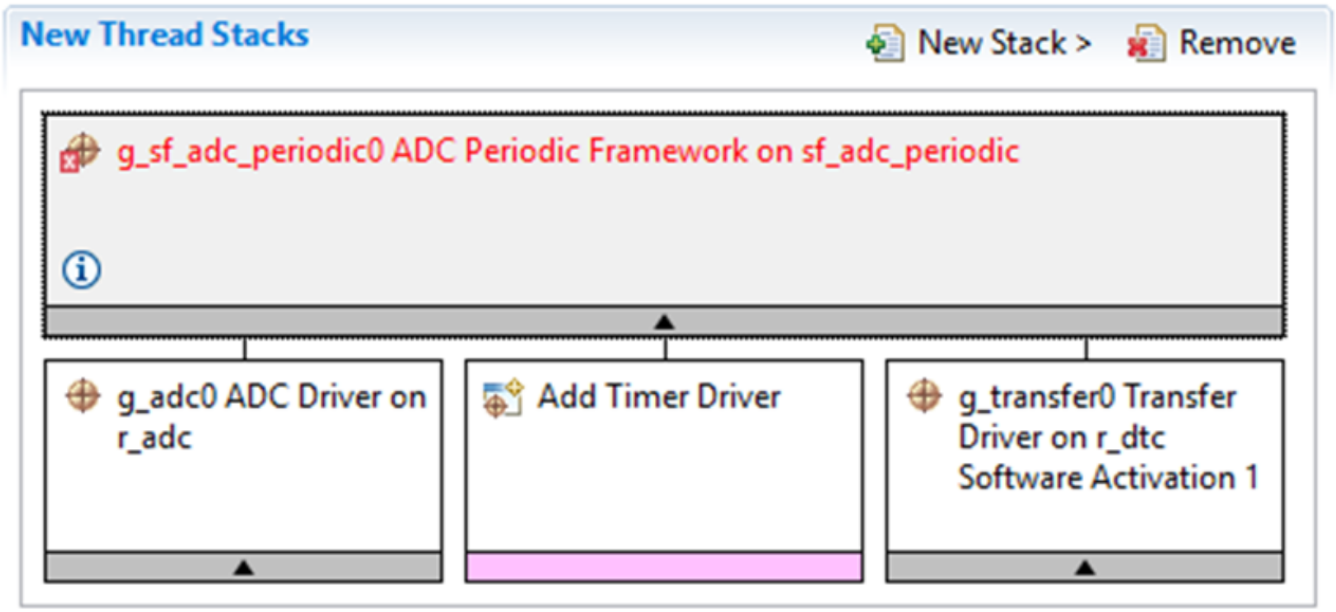


Figure 104: ADC Periodic Framework Module Stack

Note

The above diagram will have an "Add ADC Driver" block instead of the "g_adc0" block for the S1JA only.

4.1.1.5 Configuring the ADC Periodic Framework Module

The ADC Periodic Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ADC Periodic Framework Module on sf_adc_periodic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_adc_periodic0	Module name.
Name of the data-buffer to store samples	g_user_buffer	Name of the 16-bit data buffer to store samples.

Length of the data-buffer	128	Length of the buffer to which data is to be stored.
Number of sampling iterations	10	Priority of ADC Periodic Framework internal thread.
Callback	g_adc_framework_user_callback	User function that will be called once "sample_counts" number of data has been buffered.
Name of generated initialization function	sf_adc_periodic_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the ADC Periodic Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected code for parameter checking is included in the build.
Name	g_adc0	Module name.
Unit	0, 1 (S7G2 Only) Default: 0	Specify the ADC Unit to be used. The S7G2 has two units; 0 and 1.
Resolution	14-Bit (S3A7/S124 Only), 12-Bit, 10-Bit (S7G2) Default: 8-Bit (S7G2 Only)	Specify the conversion resolution for this unit.
Alignment	Right, Left Default: Right	Specify the conversion result alignment.

Clear after read	Off, On Default: On	Specify if the result register must be automatically cleared after the conversion result is read. Note: If this is enabled, then watching the result register using a debugger always results in a 0.
Mode	Single Scan	The ADC Framework preconfigures and locks this field.
Channels 0-6	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 7-10 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 11-15 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 16-20	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

Channel 21 (Unit 0 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 22 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 23-27 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Temperature Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Temperature sensor use selection for Channel Scan Mask.
Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Voltage sensor use selection for Channel Scan Mask.
Normal/Group A Trigger	ELC Event	The ADC Framework preconfigures and locks this field.
Group B Trigger (Valid Only in Group Scan Mode)	ELC Event (The only valid trigger for either group in Group Scan Mode)	The ADC Framework preconfigures and locks this field.

Group Priority (Valid only in Group Scan Mode)	Group A cannot interrupt Group B, Group A can interrupt Group B; Group B scan restarts at next trigger, Group A can interrupt Group B; Group B scan restarts immediately, Group A can interrupt Group B; Group B scan restarts immediately and scans continuously Default: Group A cannot interrupt Group B	Do not use with ADC Framework since the mode is locked to Single Scan Mode.
Add/Average Count	Disabled, Add two samples, Add three samples, Add four samples, Add sixteen samples, Average two samples, Average four samples Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask adc_channel_cfg_t::add_mask.
Channels 0-27	Disabled, Enabled Default: Disabled	This field is valid only if adc_cfg_t::add_average_count is enabled. This field determines what channels results are to be averaged or summed.
Temperature Sensor	Disabled, Enabled Default: Disabled	Temperature sensor use selection for Addition/Averaging Mask.
Voltage Sensor	Disabled, Enabled Default: Disabled	Voltage sensor use selection for Addition/Averaging Mask.
Channels 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in adc_channel_cfg_t::sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.

Sample Hold States (Applies only to the 3 channels selected above)	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if adc_channel_cfg_t::sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. Note: Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.
Callback	NULL	The ADC Framework uses the callback internally.
Scan End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Interrupt Priority selection.
Scan End Group B Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Group B Interrupt Priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SDADC HAL Module on r_sdadc(Only for S1JA)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_adc0	Module name.
Mode	Single Scan, Continuous Scan Default: Continuous Scan	In single scan mode, all channels are converted once per start trigger, and conversion stops after all enabled channels are scanned. In continuous scan mode, conversion starts after a start trigger, then continues until stopped in software.
Resolution	16 Bit, 24 Bit Default: 24 Bit	Select 24-bit or 16-bit resolution.

Alignment	Right, Left Default: Right	Select left or right alignment.
Trigger	ELC Hardware Event, Software Default: Software	Select conversion start trigger. Conversion can be started in software, or conversion can be started when a hardware event occurs if the hardware event is linked to the SDADC peripheral using the ELC API.
Vref Source	Internal, External Default: Internal	Vref can be sourced internally and output on the SBIAS pin, or Vref can be input from VREFI.
Vref Voltage	0.8 V, 1.0 V, 1.2 V, 1.4 V, 1.6 V, 1.8 V, 2.0 V, 2.2 V, 2.4 V Default: 1.0 V	Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%.
Internal Calibration During Open()	Enabled, Disabled Default: Enabled	Calibration is required for all channels configured for differential input. Internal calibration is performed automatically during open for these channels unless it is disabled here.
Callback	NULL	Enter the name of the callback function to be called when conversion completes or a scan ends.
Conversion End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Select the interrupt priority for the conversion end interrupt. [Required]
Scan End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the scan end interrupt. [Required]
Calibration End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the calibration end interrupt. [Required]

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Microseconds	See Timer Period Calculation.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: True	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.

Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Underflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	The ADC Framework preconfigures and locks this field based on channel selected in the ADC Framework.
Mode	Periodic	The ADC Framework preconfigures and locks this field.
Period Value	10	Configure timer period to trigger ADC scans.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Milliseconds	Configure units of the timer period set above.

Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	False	The ADC Framework preconfigures and locks this field.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	The ADC Framework preconfigures and locks this field.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.

Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Block	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	1	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

ADC Periodic Framework Module Pin Configuration

To access a channel, ADC channels must be set in the **Pins** tab of the ISDE. The following table illustrates the method for selecting the pins within the SSP configuration window:

Pin Selection for the ADC HAL Module on r_adc

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select **Peripherals** > **Analog: ADC > ADC0\1** > **AN_XX**

SDADC	Pins	Select **Peripherals** > **Analog: SDADC** > **SDADC0** > **AN_XX**

Note

In the cases of the internal temperature sensor and the internal voltage sensor, there are no pin configurations required.

4.1.1.6 Using the ADC Periodic Framework Module in an Application

The steps in using the ADC Periodic Framework module on sf_audio_record_adc in a typical application are:

1. Initialize the ADC using the sf_adc_periodic_api_t::open API.
2. Start the Scan of channels using the sf_adc_periodic_api_t::start API.
3. Stop the scan with the sf_adc_periodic_api_t::stop API.
4. Read the results of the conversion using the callback in application code.
5. Close the instance using the sf_adc_periodic_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

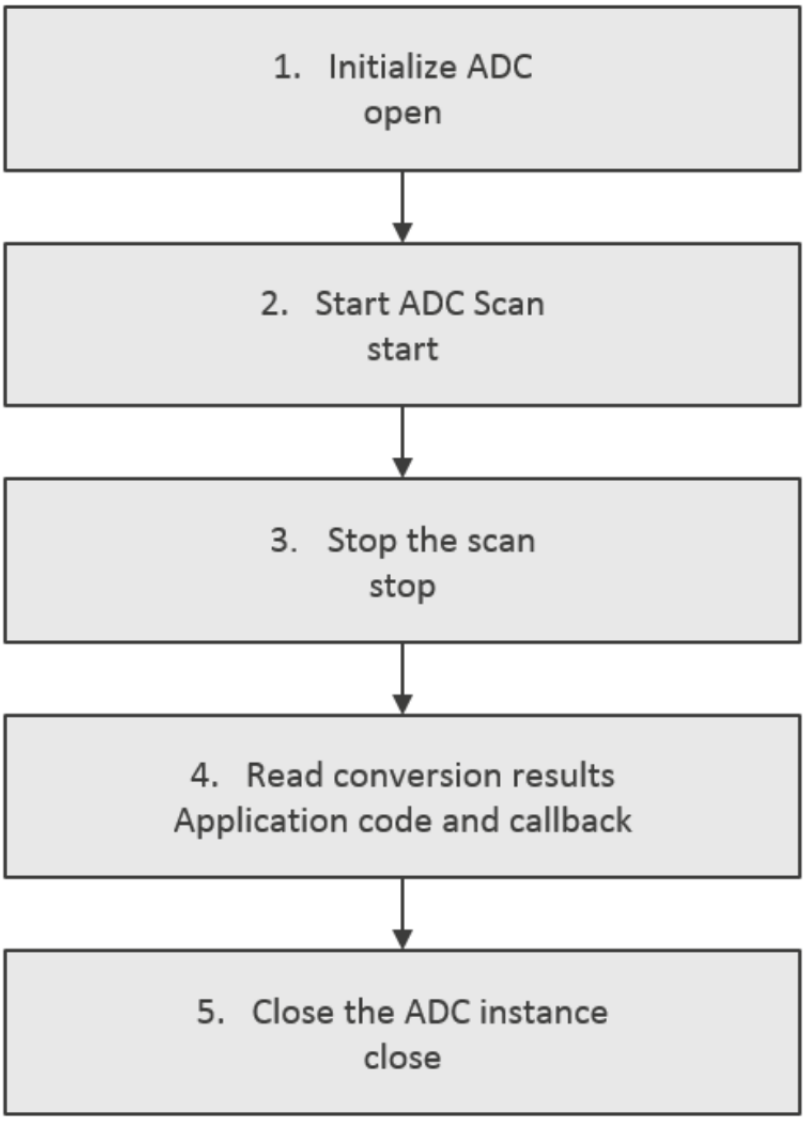


Figure 105: Flow Diagram of a Typical ADC/SDADC Periodic Framework Module Application

4.1.2 Audio Playback Framework

4.1.2.1 Audio Playback Framework Introduction

The Audio Playback Framework module provides a high-level API for audio playback applications and is implemented on either sf_audio_playback_hw_dac or sf_audio_playback_hw_i2s. The Audio Playback Framework handles the synchronization needed to play 16-bit and 8-bit pulse-code modulation (PCM) samples and uses the DAC (DAC12 or DAC8) or I2S, timer (AGT or GPT) and data-transfer (DMA or DTC) peripherals on a Synergy MCU. A user-defined callback can be created to respond to the need for additional data.

Audio Playback Framework Module Features

- Plays long buffers by splitting the data into manageable chunks.
- Repeats playback until a ThreadX timeout (for repeated audio like sine wave tones or looped background music).
- Requests next data using callback after last buffer playback begins.
- Software volume control.
- Pause and resume functions.
- Scaling, for example to move signed 16-bit PCM data into range of the unsigned 12-bit DAC or unsigned 8-bit DAC8.
- Basic mixing for multiple streams.

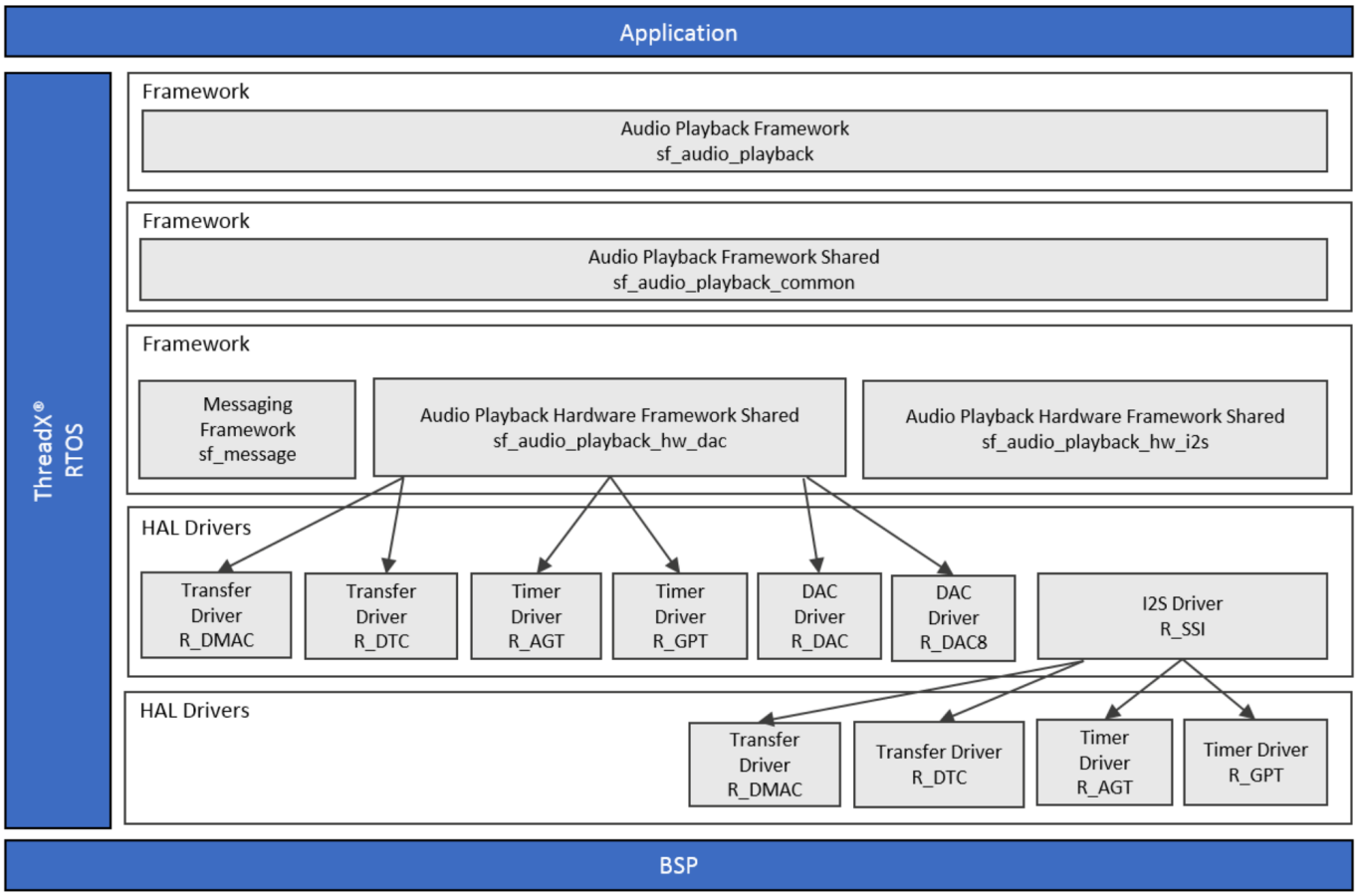


Figure 106: Audio Playback Framework Module Block Diagram

4.1.2.2 Audio Playback Framework Module APIs Overview

The Audio Playback Framework module defines APIs for operations such as opening, starting, playing and stopping. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Playback Framework Module API Summary

Function Name	Example API Call and Description

open	g_sf_audio_playback0.p_api->open(g_sf_audio_playback0.p_ctrl, g_sf_audio_playback0.p_cfg); Open a device channel for read/write and control.
start	g_sf_audio_playback0.p_api->start(g_sf_audio_playback0.p_ctrl, p_data, Timeout); Start Audio Playback Hardware.
pause	g_sf_audio_playback0.p_api->pause(g_sf_audio_playback0.p_ctrl); Pause Audio Playback Hardware.
stop	g_sf_audio_playback0.p_api->stop(g_sf_audio_playback0.p_ctrl); Stop Audio Playback Hardware.
resume	g_sf_audio_playback0.p_api->resume(g_sf_audio_playback0.p_ctrl, &buffer, length); Resume playback.
volumeSet	g_sf_audio_playback0.p_api->volumeSet(g_sf_audio_playback0.p_ctrl, volume); Sets volume.
close	g_sf_audio_playback0.p_api->close(g_sf_audio_playback0.p_ctrl); Close the audio module.
versionGet	g_sf_audio_playback0.p_api->versionGet(&version); Return the version of the module using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.2.3 Audio Playback Framework Module Operational Overview

The Audio Playback Framework module creates a thread internally to support audio playback. The following figure shows a flowchart of the audio playback framework thread and its interactions with public Audio Playback Framework API functions:

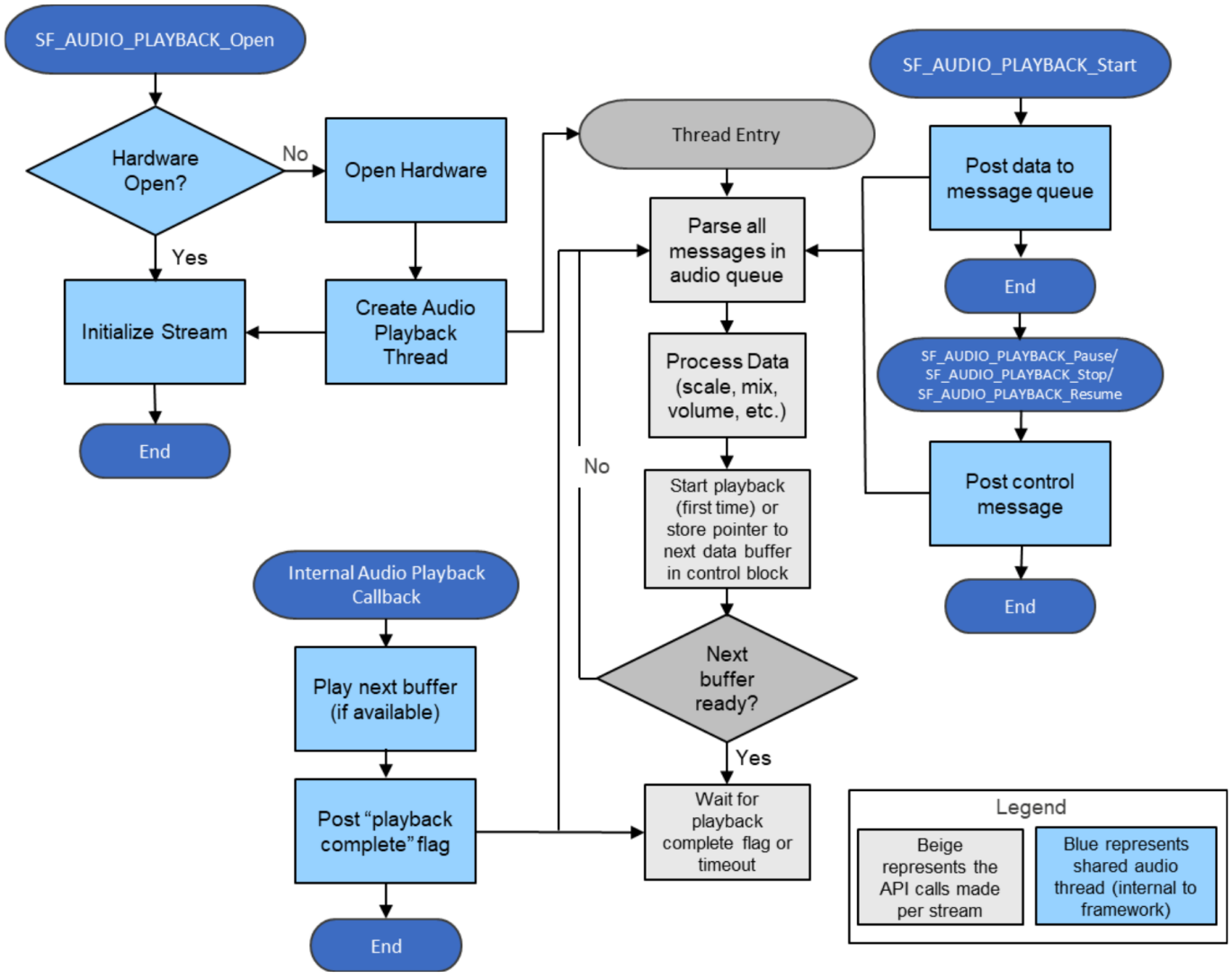


Figure 107: Audio Playback Framework Module Flow Chart

Suggested use of the audio playback framework:

- Create a semaphore (for example, g_sf_audio_playback_semaphore). This can be done on the **Threads** tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).
- Create a callback function (for example, sf_audio_playback_callback). Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the audio playback framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a

buffer from the messaging framework, then create your audio playback data structure inside the buffer.

The Audio Playback Framework supports multiple audio streams on a single hardware port. A block diagram of the modules required if two streams are used is shown in the following figure:

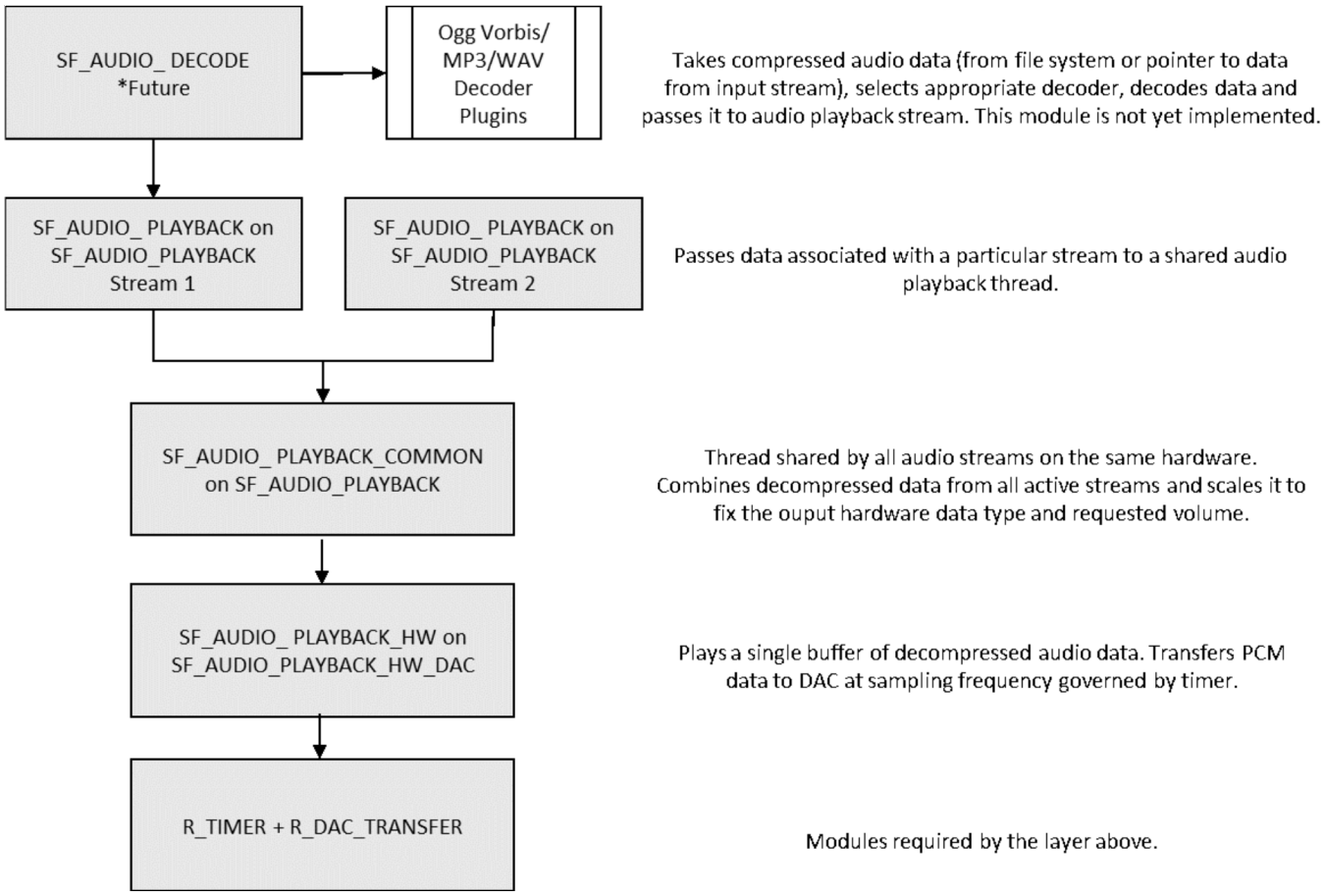


Figure 108: Audio Playback Framework Module Audio Streams

Audio Playback Framework Module Important Operational Notes and Limitations

Audio Playback Framework Module Operational Notes

Configuring Messages

Use the Messaging Framework configurator on the Messaging tab to configure the messaging framework:

1. Highlight the Audio Playback event class.
2. Add a new subscriber. Select the following configurations and make sure the Message Class Instance property set in the **Properties** tab of the Audio Playback Framework on sf_audio_playback module is between the Start and End instance.
    - Thread: Any thread in the application.
    - Start: First audio instance used in application.
    - End: Last audio instance used in application.
3. Highlight the new Subscriber in the Audio Playback Subscribers. Record the Symbol name.
4. Go back to the **Threads** tab.
5. Highlight the Audio Playback Framework Shared module in HAL/Common, and set the Audio

Message Queue Name to the Symbol name from the Audio Playback Subscriber.

Using the I2S Implementation

The audio framework I2S hardware port has dependencies on the I2S Driver module. The I2S driver module can be accelerated with DTC.

- Set the ISDE properties for the I2S driver module.
  - Set the Audio Clock Frequency (Hertz) to the frequency of the input audio clock used.
  - Set the Sampling Frequency (Hertz) to the sampling frequency of your audio data.
  - Set the Data Bits and Word Length to 16 bits (audio framework accepts 16 bit samples only).
  - Enable the SSIn TXI and SSIn INT interrupts.
- The Transfer module on r_dtc is added automatically.

Using the DAC Implementation

The audio framework DAC hardware port has dependencies on the Timer, DAC, and Transfer API modules.

- Add a Timer module.
  - Set the Frequency in Hz to the sampling frequency of your audio data.
  - Enable the interrupt if using DTC as the transfer module (recommended).
- Add a DAC (DAC12 or DAC8) module.
- Add a Transfer module on r_dtc.
  - Set Destination pointer to &R_DAC->DADRn[0] if using DAC channel 0 or &R_DAC->DADRn[1] if using DAC channel 1.
  - Set Destination pointer to &R_DAC8->DADRn[2] if using DAC8 channel 2 (S128) or &R_DAC8->DADRn[0] if using DAC8 channel 0 (S1JA).
  - Set the Activation source to the timer interrupt chosen above.

Other Operational Notes

The Queue used must match the name specified in Properties for Audio Playback Framework Shared on sf_audio_playback (default is g_sf_audio_playback_queue).

Audio Playback Framework Module Limitations

- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.1.2.4 Including the Audio Playback Framework Module in an Application

This section describes how to include the Audio Playback Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Playback Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Audio Playback Framework module is g_sf_audio_playback0. This name can be changed in the

associated Properties window.)

Audio Playback Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback Audio Playback Framework on g_sf_audio_playback	Threads	**New Stack> Framework> Audio> Audio Playback Framework on g_sf_audio_playback**

When the Audio Playback Framework module on sf_audio_playback is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

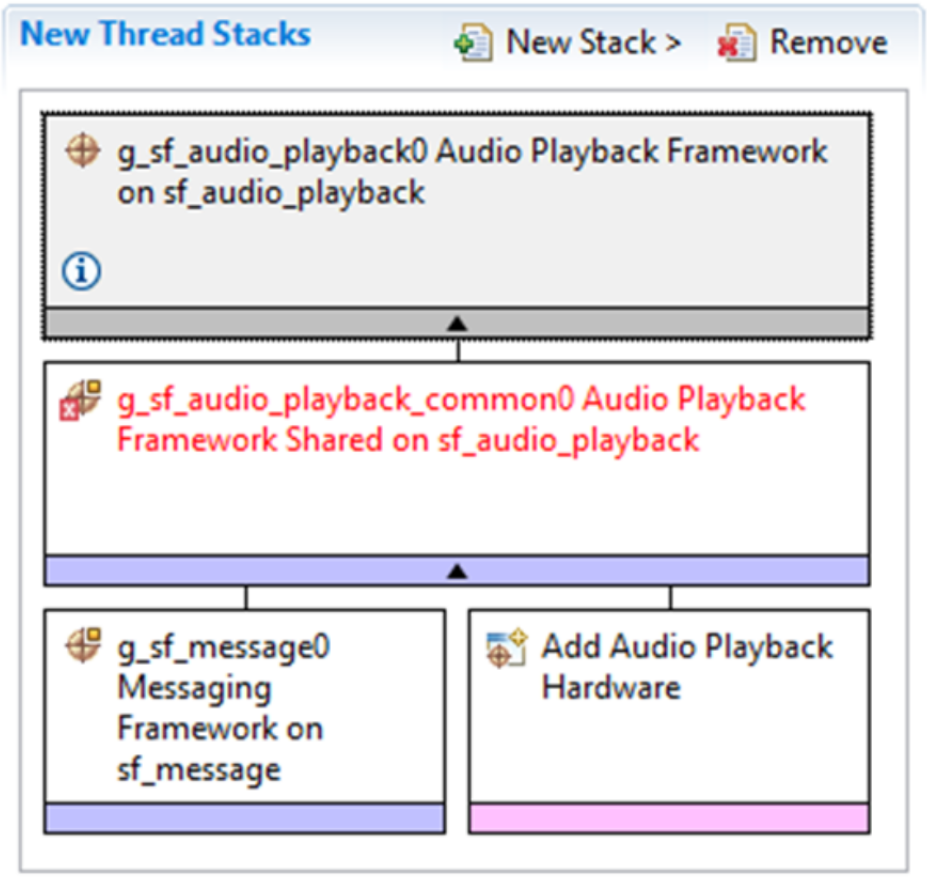


Figure 109: Audio Playback Framework Module Stack

4.1.2.5 Configuring the Audio Playback Framework Module

The Audio Playback Framework module must be configured by the user for the desired operation.

The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Audio Playback Framework Module on sf_audio_playback

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Buffer Size Bytes	512	Buffer size bytes selection.
Maximum Number of Streams	1	Maximum number of streams selection.
Thread Stack Size	512	Thread stack size selection.
Name	g_sf_audio_playback0	Module name.
Message Class Instance	0	Message class instance selection.
Callback	NULL	Callback selection.
Name of generated initialization function	sf_audio_playback_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

Increasing the buffer size will increase the RAM consumption of this module.
The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select the DAC Channel based on the target hardware implementation. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Audio Playback Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the Audio Playback Framework Shared on sf_audio_playback

ISDE Property	Value	Description
Name	g_sf_audio_playback_common0	Module name.
Thread Priority	3	Thread priority selection.
Audio Message Queue Name	g_sf_audio_playback_queue	Audio message queue name selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Messaging Framework on sf_message

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Message Queue Depth (Total number of messages to be enqueued in a Message Queue)	16	Specify the size of Thread X Message Queue in bytes for Message Subscribers. This value is applied to all the Message Queues.
Name	g_sf_message0	The name of Messaging Framework module control block instance.
Work memory size in bytes	2048	Specify the work memory size in bytes. Choosing a small number results in a small number of buffers, which can be allocated at the same time (you can confirm the total buffer number on sf_message_instance_ctrl_t::number_of_buffers). If the value is smaller than the peak number of messages to be posted at the same time, the Framework has a buffer allocation failure, affecting system performance.

Pointer to subscriber list array	p_subscriber_lists	Specify the name of pointer to the Subscriber List array.
name of the block pool internally used in the messaging framework	sf_msg_blk_pool	The name of memory block memory the Framework creates in the control block. This parameter might be useful for debugging purpose but NULL can be specified for saving memory.
Name of generated initialization function	sf_message_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Audio Playback Hardware Framework Shared on sf_audio_playback_hw_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
DMAC Support	Disabled, Enabled Default: Disabled	DMAC support selection.
Name	g_sf_audio_playback_hw0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer0	Module name.
Channel	0	Channel selection.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.

Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source	Software Activation, Peripheral Events Default: Software Activation	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dmac Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.

Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.

Period Unit	Hertz	See Timer Period Calculation.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Hertz	See Timer Period Calculation.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.

GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DAC HAL Module on r_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac0	Module name.
Channel	0	Set to 0 for output DA0 or 1 for output DA1.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false if power supply interference between the analog modules is not a problem, or if asynchronous conversion by the DAC module is desired.

Data Format	Right Justified	Set to zero, if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.
Output Amplifier	Enable, Disable Default: Disable	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Audio Playback Framework Shared on sf_audio_playback_hw_i2s

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_sf_audio_playback_hw0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of (sampling_freq_hz * word_length_in_bits).
Sampling Frequecy (Hertz)	44100	Sampling frequency of audio data.
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.

Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Transmit interrupt priority selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Receive interrupt priority selection.
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Idle/error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.
Period Unit	Hertz	Period unit selection.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	FALSE	Auto start selection.
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection.
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection.
Callback	NULL	Callback selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Audio Playback Framework Module Clock Configuration

The Audio Playback Framework hardware modules use the peripheral clocks available in the clock configuration window.

Audio Playback Framework Module Pin Configuration

The DAC or SSI peripheral module uses pins on the MCU to communicate with external devices. I/O pins must be selected and configured as required by the external device. The following tables illustrate the method for selecting the pins within the SSP configuration window, and show an example selection for the associated pins.

Pin Selection Sequence for the Audio Playback Framework Module

Resource	ISDE Tab	Pin selection Sequence
DAC	Pins	Select **Peripherals** > **Analog:DAC12** > **DAC120/121**
SSI	Pins	Select **Peripherals** > **Connectivity:SSI** > **SSI/0/1**

Note

The selection sequence assumes the DAC0/DAC1 or the SSI0/SSI1 is the desired hardware target of the driver.

Pin Configuration Settings for the DAC Driver on r_dac

Pin Configuration Property	Value	Description
Operation Mode	Enabled, Disabled	Operation selection.
DAC	None, P014 (Default: P014)	DAC pin selection.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.2.6 Using the Audio Playback Framework Module in an Application

The typical steps in using the Audio Playback Framework module in an application are:

1. Initialize an audio stream using the sf_audio_playback_api_t::open API.
2. Use the callback function to post to a semaphore with an initial count equal to the number of buffers required. This is implemented in the application code.
   - *Note*

     Get this semaphore in the application thread before calling sf_audio_playback_api_t::start API.
3. Acquire a buffer from the Messaging Framework using sf_message_api_t::bufferAcquire API.
   - *Note*

     Create the Audio Framework Data Structure using sf_audio_playback_data_t inside the buffer.
4. Start the Audio Playback Framework using the sf_audio_playback_api_t::start API.

Note

If multiple streams are desired, repeat steps 1-4 for any additional audio streams. A separate audio stream should only be used if the streams need to play simultaneously using mixing. If audio sounds are always played in

sequence and never overlap, the stream can be reused.

These common steps are illustrated in a typical operational flow in the following figure:

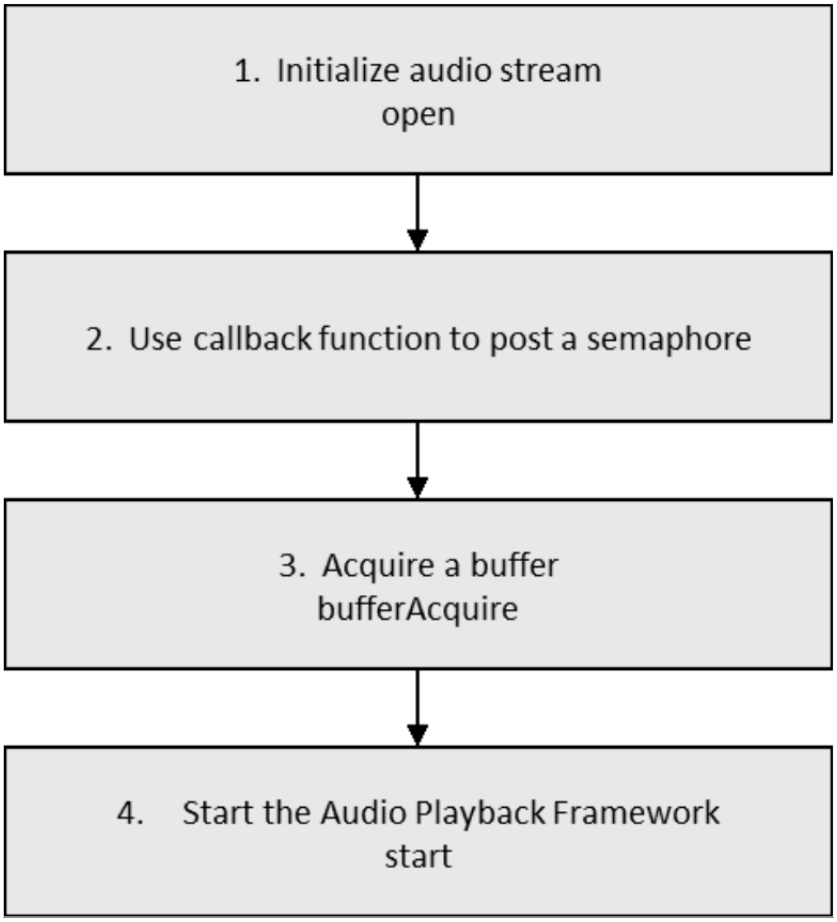


Figure 110: Flow Diagram of a Typical Audio Playback Framework Module Application

4.1.3 Audio Playback Hardware Framework Shared on sf_audio_playback_hw_dac

4.1.3.1 Audio Playback DAC Framework Introduction

The Audio Playback Framework DAC module provides a high-level API for audio playback applications and handles the synchronization needed to play 8-bit or 16-bit pulse-code modulation (PCM) samples. The Audio Playback DAC Framework uses the DAC/DAC8, timer (AGT or GPT) and data-transfer (DMA or DTC) peripherals on a Synergy MCU. A user-defined callback can be created to respond to the need for additional data.

Audio Playback DAC Framework Module Features

- Plays long buffers by splitting the data into manageable chunks.
- Repeats playback until a ThreadX timeout (for repeated audio like sine wave tones or

looped background music).
- Requests next data using callback after last buffer playback begins.
- Software volume control.
- Pauses and resumes functions.
- Scaling, for example, to move signed 16-bit PCM data into range of the unsigned 12-bit or 8-bit DAC.
- Basic mixing for multiple streams.

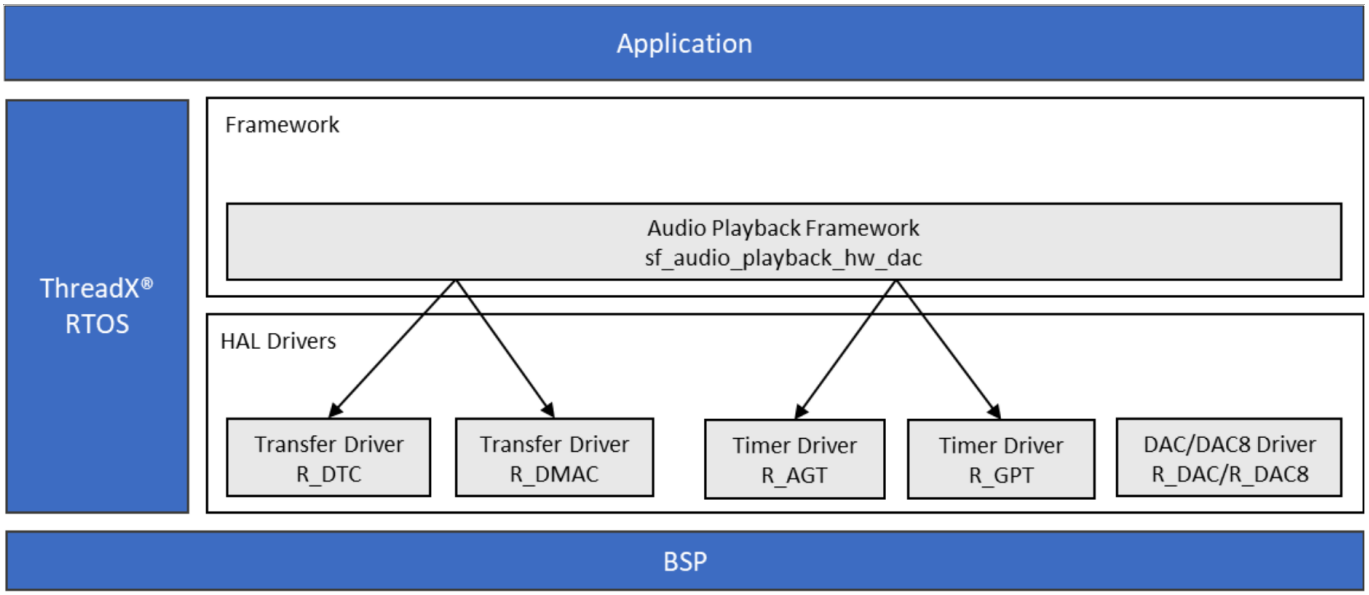


Figure 111: Audio Playback DAC Framework Module Block Diagram

Note

Selection options of DAC or DAC8 Driver is MCU specific.

4.1.3.2 Audio Playback DAC Framework Module APIs Overview

The Audio Playback DAC Framework module defines APIs for operations such as opening, starting, playing and stopping. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Playback DAC Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_audio_playback_hw0.p_api->open(g_sf_audio_playback_hw0.p_ctrl, g_sf_audio_playback_hw0.p_cfg); Open a device channel for read/write and control.
start	g_sf_audio_playback_hw0.p_api->start(g_sf_audio_playback_hw0.p_ctrl, &p_data, Timeout); Start Audio Playback Hardware.

stop	g_sf_audio_playback_hw0.p_api->stop(g_sf_audio_playback_hw0.p_ctrl); Stop Audio Playback Hardware.
play	g_sf_audio_playback_hw0.p_api->play (g_sf_audio_playback_hw0.p_ctrl, p_buffer, length); Play audio buffer.
dataTypeGet	g_sf_audio_playback_hw0.p_api->dataTypeGet(g_sf_audio_playback_hw0.p_ctrl, p_data_type); Store expected data type in provided pointer p_data_type.
close	g_sf_audio_playback_hw0.p_api->close(g_sf_audio_playback_hw0.p_ctrl); Close the audio module.
versionGet	g_sf_audio_playback_hw0.p_api->versionGet(&version); Return the version of the module with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.3.3 Audio Playback DAC Framework Module Operational Overview

The Audio Playback Framework DAC module creates a thread internally to support audio playback. The following figure shows a flowchart of the Audio Playback Framework thread and its interactions with public Audio Playback Framework APIs.

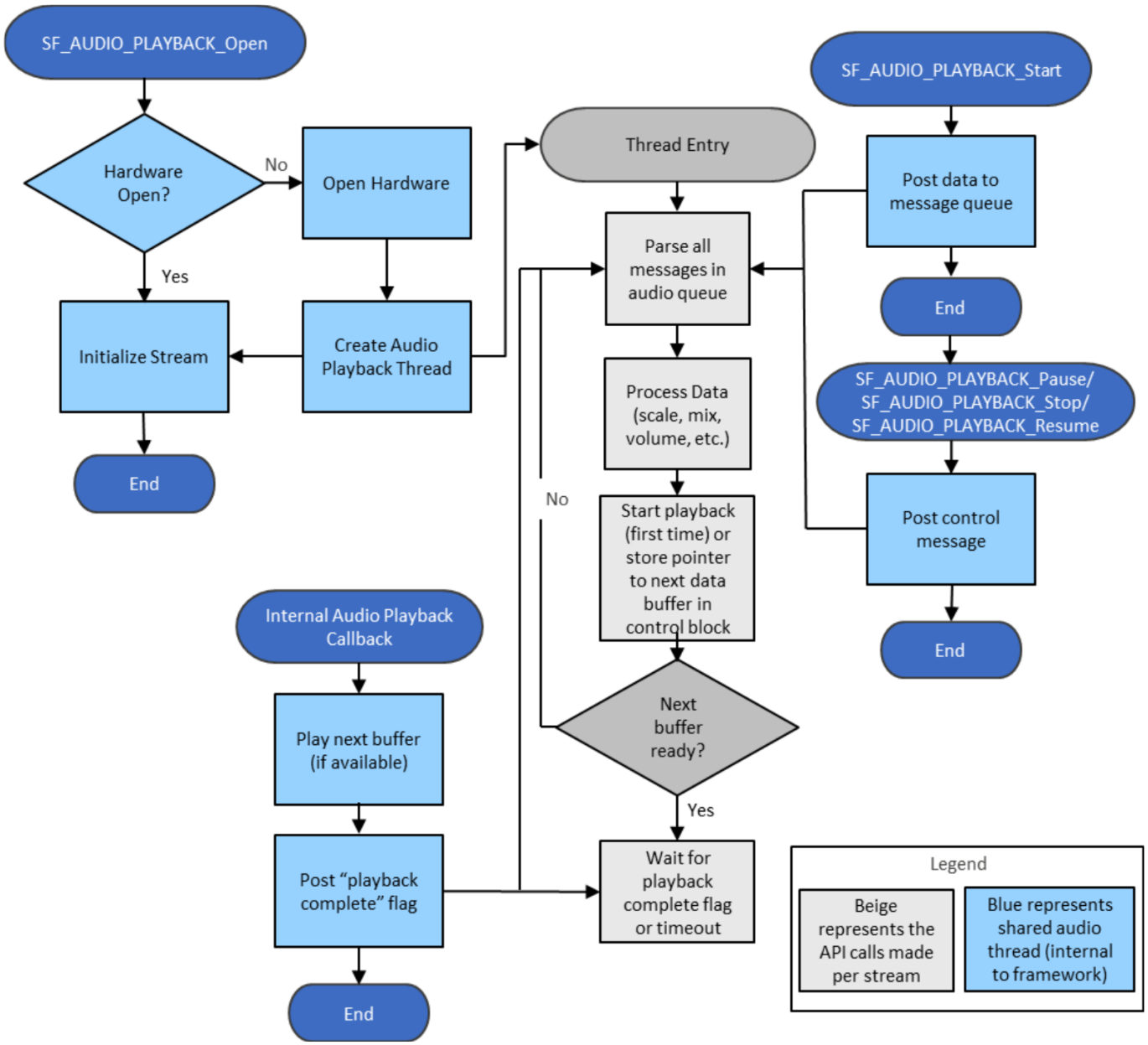


Figure 112: Audio Playback DAC Framework Module Flow Chart

Suggested use of the Audio Playback Framework:

- Create a semaphore (for example, 'g_sf_audio_playback_semaphore'). This can be done on the **Threads** tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).
- Create a callback function (for example, 'sf_audio_playback_callback)'. Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the Audio Playback Framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a buffer from the messaging framework, then create your audio playback data structure inside the buffer.

The Audio Playback DAC Framework supports multiple audio streams on a single hardware port. A

block diagram of the modules required if two streams are used is shown in the following figure:

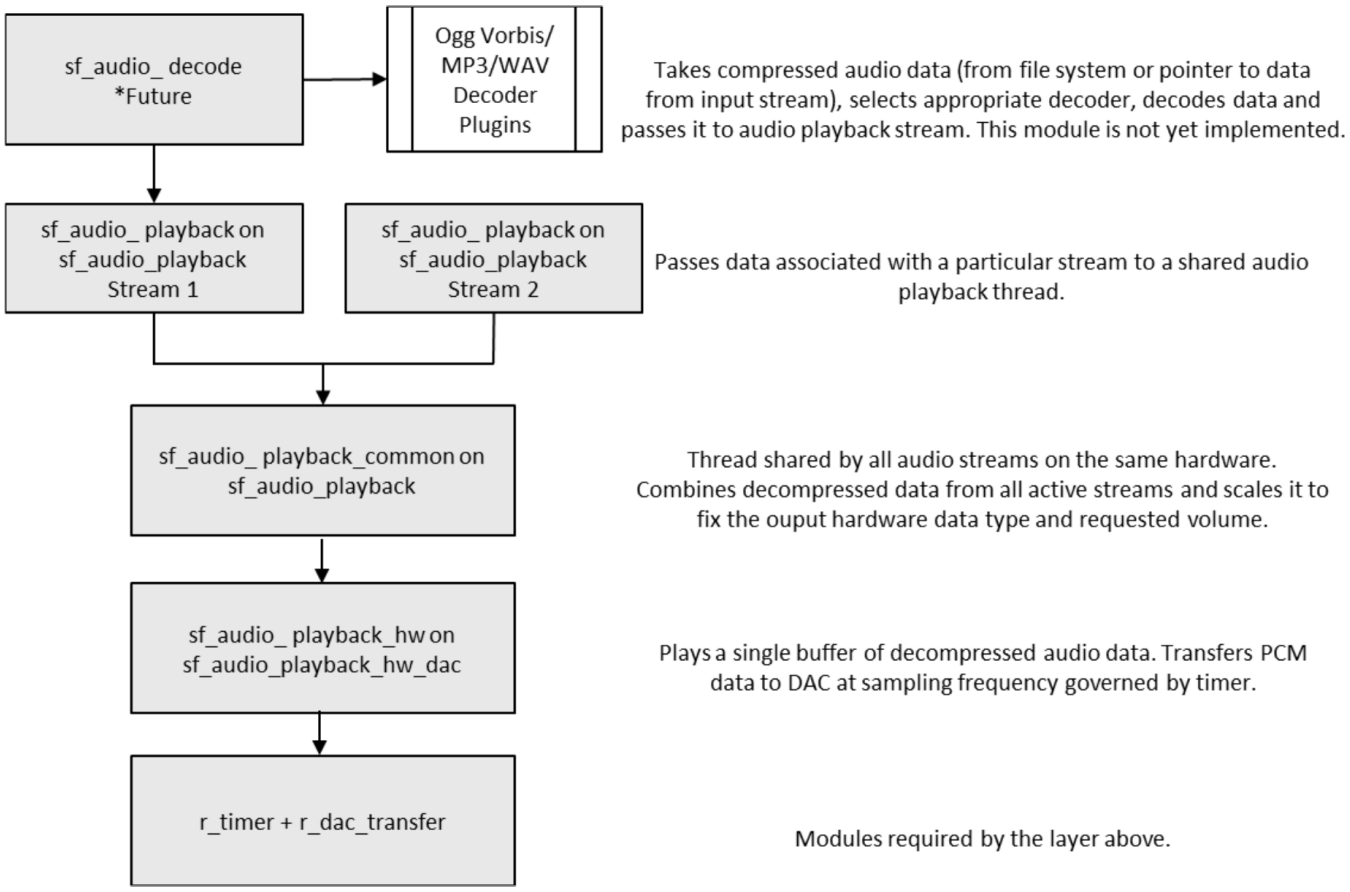


Figure 113: Audio Playback DAC Framework Implementing Multiple Audio Streams

Audio Playback DAC Framework Module Important Operational Notes and Limitations

Audio Playback DAC Framework Module Operational Notes

- The Audio Framework DAC hardware port uses the transfer API to transfer audio data from a playback buffer to DAC at a sampling frequency defined by an internal timer.
- The Audio Framework DAC hardware port has dependencies on the Timer, DAC/DAC8, and Transfer API modules.
- Add a Timer module.
    - Set the Frequency in Hz to the sampling frequency of your audio data.
    - Enable the interrupt if using DTC as the transfer module (recommended).
- Add a Transfer module: select either the DTC or DMAC.
- For DTC:
    - Set Destination pointer to &R_DAC->DADRn[0] if using DAC channel 0 or &R_DAC->DADRn[1] if using DAC channel 1.
    - Set Destination pointer to &R_DAC8->DADRn[2] if using DAC8 channel 2 (S128) or &R_DAC8->DADRn[0] if using DAC8 channel 0 (S1JA).
    - Set the activation source to the timer interrupt chosen above.
- For DMAC:
    - Enable the DMAC support
    - Set Destination pointer to &R_DAC->DADRn[0] if using DAC channel 0 or &R_DAC->DADRn[1] if using DAC channel 1.
    - Set Destination pointer to &R_DAC8->DADRn[2] if using DAC8 channel 2 (S128) or

&R_DAC8->DADRn[0] if using DAC8 channel 0 (S1JA).
    - Set the activation source to the timer interrupt chosen above.
- The Audio Playback DAC Framework is designed to support the following MCU families with no changes to the API:
    - S7G2
    - S3A3
    - S5D9
    - S3A7
    - S124
    - S3A6
    - S5D5
    - S3A1
    - S128
    - S1JA
    - S5D3

Audio Playback DAC Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.3.4 Including the Audio Playback DAC Framework Module in an Application

This section describes how to include the Audio Playback DAC Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Playback DAC Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Audio Playback DAC Framework module is g_sf_audio_playback_hw0. This name can be changed in the associated Properties window.)

Audio Playback DAC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback_hw0 Audio Playback Hardware Framework on g_sf_audio_playback_hw0	Threads	**New Stack> Framework> Audio> Audio Playback Hardware Framework on g_sf_audio_playback_hw_dac**

When the Audio Playback DAC Framework module on sf_audio_playback_hw_dac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

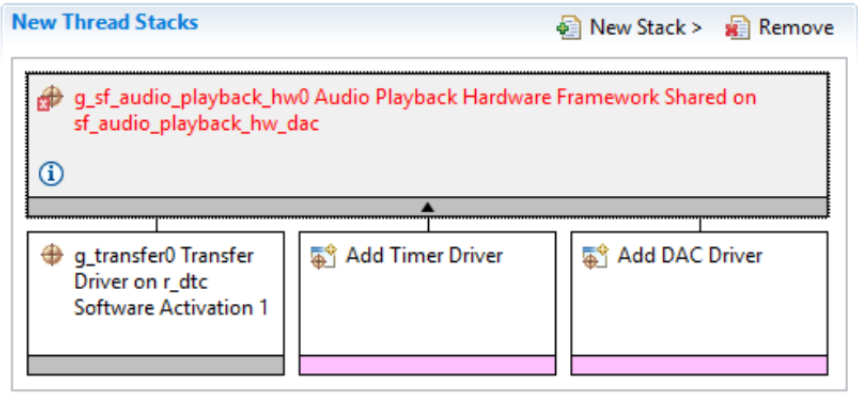


Figure 114: Audio Playback DAC Framework Module Stack

Note

Selection options of DAC or DAC8 Driver is MCU specific

4.1.3.5 Configuring the Audio Playback DAC Framework Module

The Audio Playback DAC Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Audio Playback DAC Framework Module on sf_audio_playback_hw_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
DMAC Support	Disabled, Enabled Default: Disabled	DMAC support selection.
Name	g_sf_audio_playback_hw0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select the DAC channel based on the target hardware implementation. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Audio Playback DAC Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer0	Module name.
Channel	0	Channel selection.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source	Software Activation, Peripheral Events Default: Software Activation	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback	NULL	Callback selection.

Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dmac Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Hertz	See Timer Period Calculation.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.

Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.

Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Hertz	See Timer Period Calculation.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.

Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DAC HAL Module on r_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac0	Module name.
Channel	0	Set to 0 for output DA0 or 1 for output DA1.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false if power supply interference between the analog modules is not a problem, or if asynchronous conversion by the DAC Module is desired.
Data Format	Right Justified	Set to zero, if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.

Output Amplifier	Enable, Disable Default: Disable	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DAC8 HAL Module r_dac8

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac8_0	Module name.
Channel	0	Channel selection.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Choose whether to sync with the ADC module.
Data Format	Right Justified	Data format selection.
DAC Mode	Normal Mode, Real-time (Event Link) Mode Default: Normal Mode	DAC mode selection.
Charge Pump Enabled (Requires MOCO active)	Enabled, Disabled Default: Enabled	Enable or disable the charge pump.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Audio Playback DAC Framework Module Clock Configuration

The Audio Playback DAC Framework hardware modules use the peripheral clocks available in the Clocks configuration window.

Audio Playback DAC Framework Module Pin Configuration

The DAC or SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following tables illustrate the method for selecting the pins within the SSP configuration window and show an example selection for the associated pins.

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for the Audio Playback DAC Framework Module

Resource	ISDE Tab	Pin selection Sequence

DAC	Pins	Select **Peripherals** > **Analog:DAC12** > **DAC120/121**
SSI	Pins	Select **Peripherals** > **Connectivity:SSI** > **SSI/0/1**

Note

The selection sequence assumes the DAC0/DAC1 or the SSI0/SSI1 is the desired hardware target of the driver.

Pin Configuration Settings for the DAC HAL Module on r_dac

Pin Configuration Property	Value	Description
Operation Mode	Enabled, Disabled	Operation selection.
DAC	None, P014 Default: P014	DAC pin selection.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.3.6 Using the Audio Playback DAC Framework Module in an Application

The typical steps in using the Audio Playback DAC Framework module in an application are:

1. Initialize the Audio Playback DAC Framework using the sf_audio_playback_hw_api_t::open API
2. Start the low level hardware of the Audio Playback DAC Framework using the sf_audio_playback_hw_api_t::start API.
3. Play the PCM audio samples using the sf_audio_playback_hw_api_t::play API.
4. PCM audio samples supported by the hardware is provided by the sf_audio_playback_hw_api_t::dataTypeGet API.
5. Stop the low level hardware of the Audio Playback DAC Framework using sf_audio_playback_hw_api_t::stop API.
6. Close the Audio Playback DAC Framework using the sf_audio_playback_hw_api_t::close API.

These common steps are illustrated in a typical operational flow in the following figure:

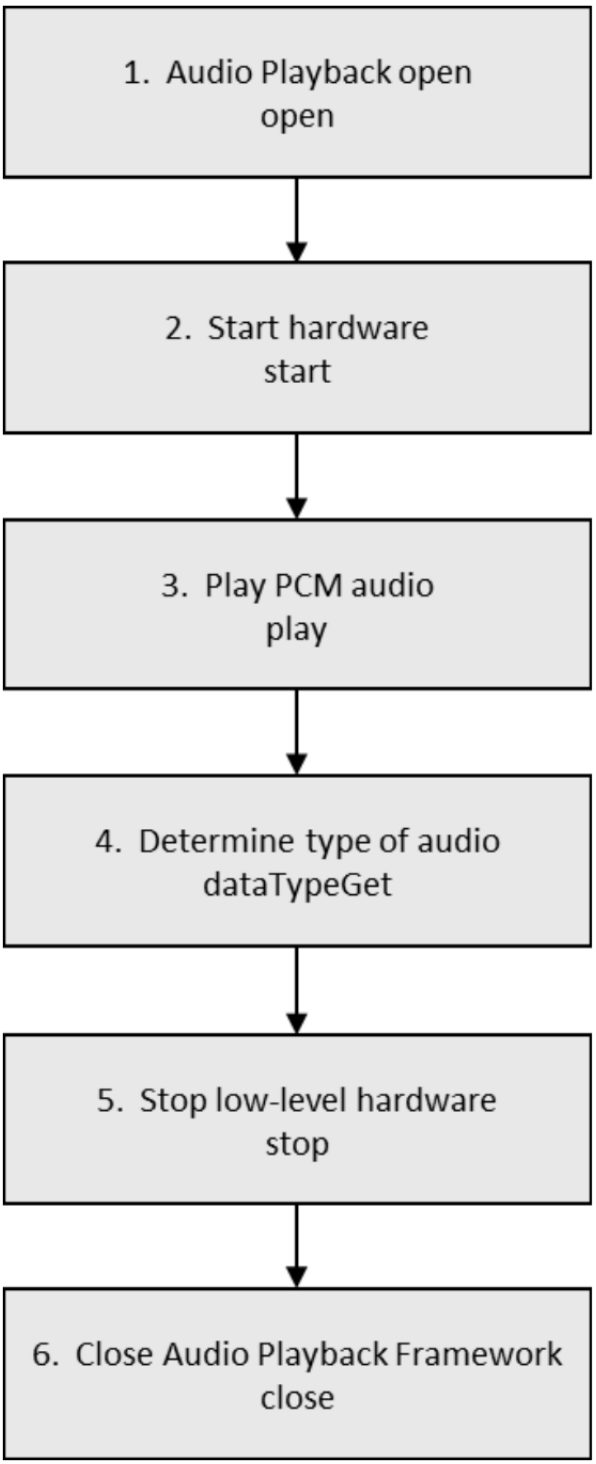


Figure 115: Flow Diagram of a Typical Audio Playback DAC Framework Module Application

4.1.4 Audio Playback Hardware Framework Shared on sf_audio_playback_hw_i2s

4.1.4.1 Audio Playback I2S Framework Introduction

The I2S Audio Playback Framework module provides a high-level API for Audio Playback applications and handles the synchronization needed to play 8-bit or 16-bit pulse-code modulation (PCM) samples. The Audio Playback Framework uses the I2S, Timer (AGT or GPT) and Data Transfer (DMA or DTC) peripherals on a Synergy MCU. A user defined callback can be created to respond to the need for additional data.

Audio Playback I2S Framework Module Features

- Plays long buffers by splitting the data into manageable chunks.
- Repeats playback until ThreadX timeout (for repeated audio like sine wave tones or looped background music).
- Requests next data using callback after last buffer playback begins.
- Software volume control.
- Pauses and resumes functions.
- Basic mixing for multiple streams.

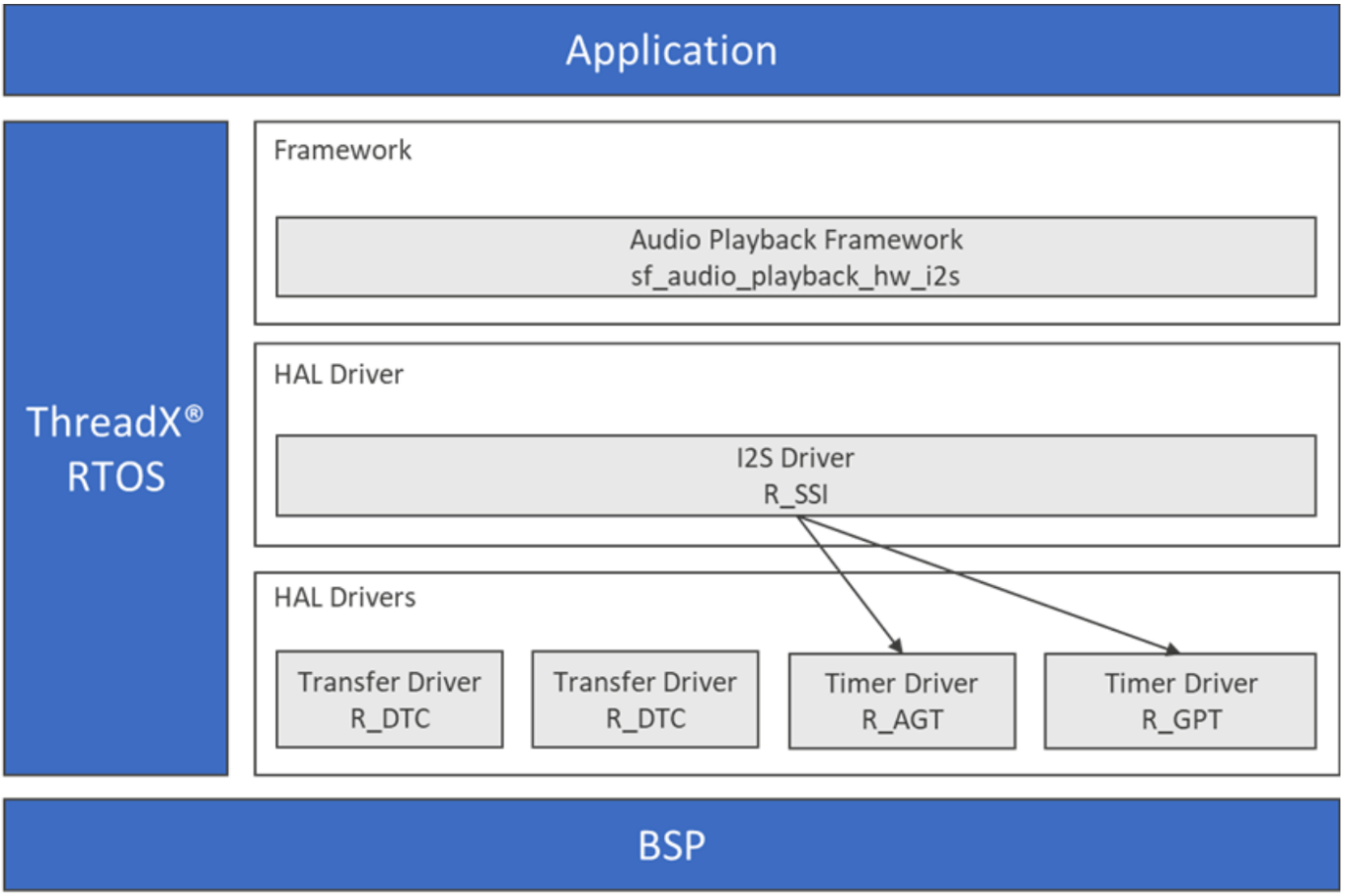


Figure 116: Audio Playback I2S Framework Module Block Diagram

4.1.4.2 Audio Playback I2S Framework Module APIs Overview

The Audio Playback I2S Framework module defines APIs for operations such as opening, starting, playing and stopping. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Playback I2S Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_audio_playback_hw0.p_api->open(g_sf_audio_playback_hw0.p_ctrl, g_sf_audio_playback_hw0.p_cfg); Open a device channel for read/write and control.
start	g_sf_audio_playback_hw0.p_api->start(g_sf_audio_playback_hw0.p_ctrl, &p_data, Timeout); Start Audio Playback Hardware.
stop	g_sf_audio_playback_hw0.p_api->stop(g_sf_audio_playback_hw0.p_ctrl); Stop Audio Playback Hardware.
play	g_sf_audio_playback_hw0.p_api->play (g_sf_audio_playback_hw0.p_ctrl, p_buffer, length); Play audio buffer.
dataTypeGet	g_sf_audio_playback_hw0.p_api->dataTypeGet(g_sf_audio_playback_hw0.p_ctrl, p_data_type); Stores expected data type in provided pointer p_data_type.
close	g_sf_audio_playback_hw0.p_api->close(g_sf_audio_playback_hw0.p_ctrl); Close the audio module.
versionGet	g_sf_audio_playback_hw0.p_api->versionGet(&version); Return the version of the module with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If thisnumber is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.4.3 Audio Playback I2S Framework Module Operational Overview

The I2S Audio Playback Framework module creates a thread internally to support audio playback. The figure below shows a flowchart of the audio playback framework thread and its interactions with public Audio Playback Framework APIs.

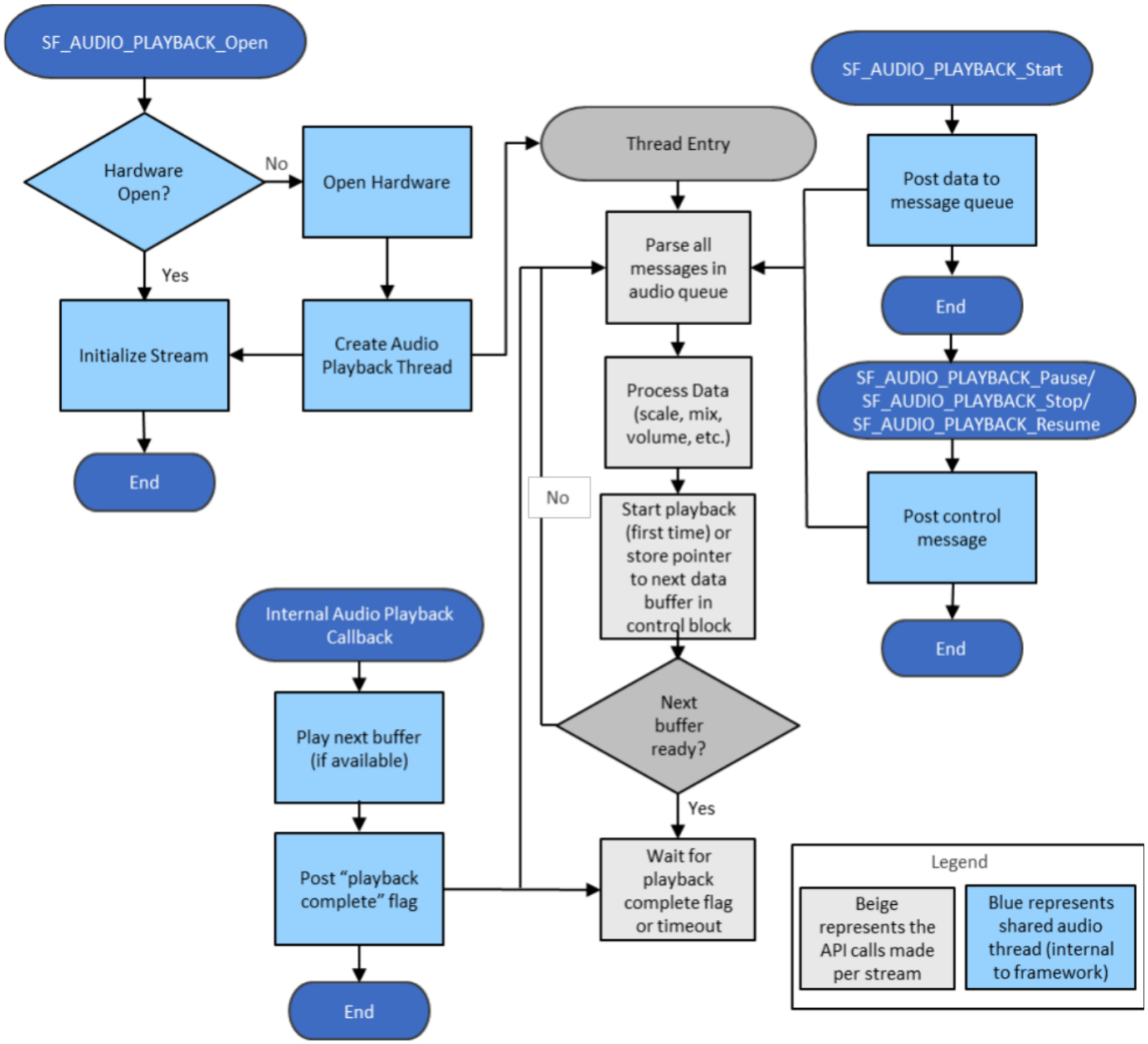


Figure 117: Audio Playback DAC Framework Module Flow Chart

Suggested use of the audio playback framework:

- Create a semaphore (for example g_sf_audio_playback_semaphore). This can be done on the **Threads** tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).

- Create a callback function (for example sf_audio_playback_callback). Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the audio playback framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a buffer from the messaging framework, then create your audio playback data structure inside the buffer.

The Audio Playback Framework supports multiple audio streams on a single hardware port. A block diagram of the modules required if two streams are used is shown in following figure:

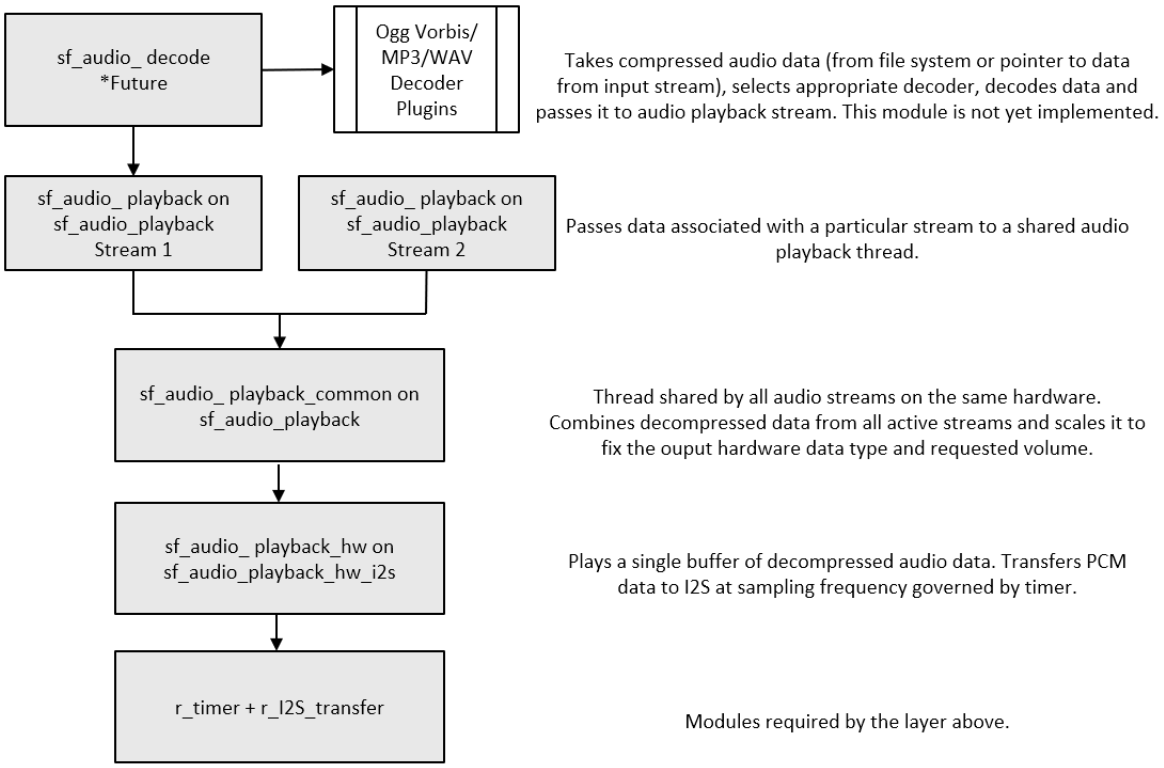


Figure 118: Audio Playback DAC Framework Implementing Multiple Audio Streams

Audio Playback I2S Framework Module Important Operational Notes and Limitations

Audio Playback I2S Framework Module Operational Notes

- The queue used must match the name specified in Properties for Audio Playback Framework Shared on sf_audio_playback (default is g_sf_audio_playback_queue).
- The audio framework I2S hardware port has dependencies on the I2S driver module. The I2S driver module can be accelerated with DTC (recommended).
- I2S driver module.
    - Set the Audio Clock Frequency (Hertz) to the frequency of the input audio clock used.
    - Set the Sampling Frequency (Hertz) to the sampling frequency of your audio data.
    - Set the Data Bits and Word Length to 16 bits (audio framework accepts 16 bit samples only).
    - Enable the SSIn TXI and SSIn INT interrupts.
- Transfer module on r_dtc (recommended).
- Set the activation source to the SSIn TXI interrupt.

- The Audio Playback I2S Framework is designed to support the following MCU families with no changes to the API:
  - S7G2
  - S3A7
  - S5D9
  - S3A3
  - S3A6
  - S5D5
  - S3A1
  - S5D3

Audio Playback I2S Framework Module Limitations

- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.1.4.4 Including the Audio Playback I2S Framework Module in an Application

This section describes how to include the Audio Playback I2S Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Playback I2S Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Audio Playback I2S Framework module is g_sf_audio_playback_hw0. This name can be changed in the associated Properties window.)

Audio Playback I2S Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback_hw0 Audio Playback Hardware Framework on sf_audio_playback_hw_i2s	Threads	**New Stack> Framework> Audio> Audio Playback Hardware Framework on sf_audio_playback_hw_i2s**

When the Audio Playback I2S Framework module on sf_audio_playback_hw_i2s is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

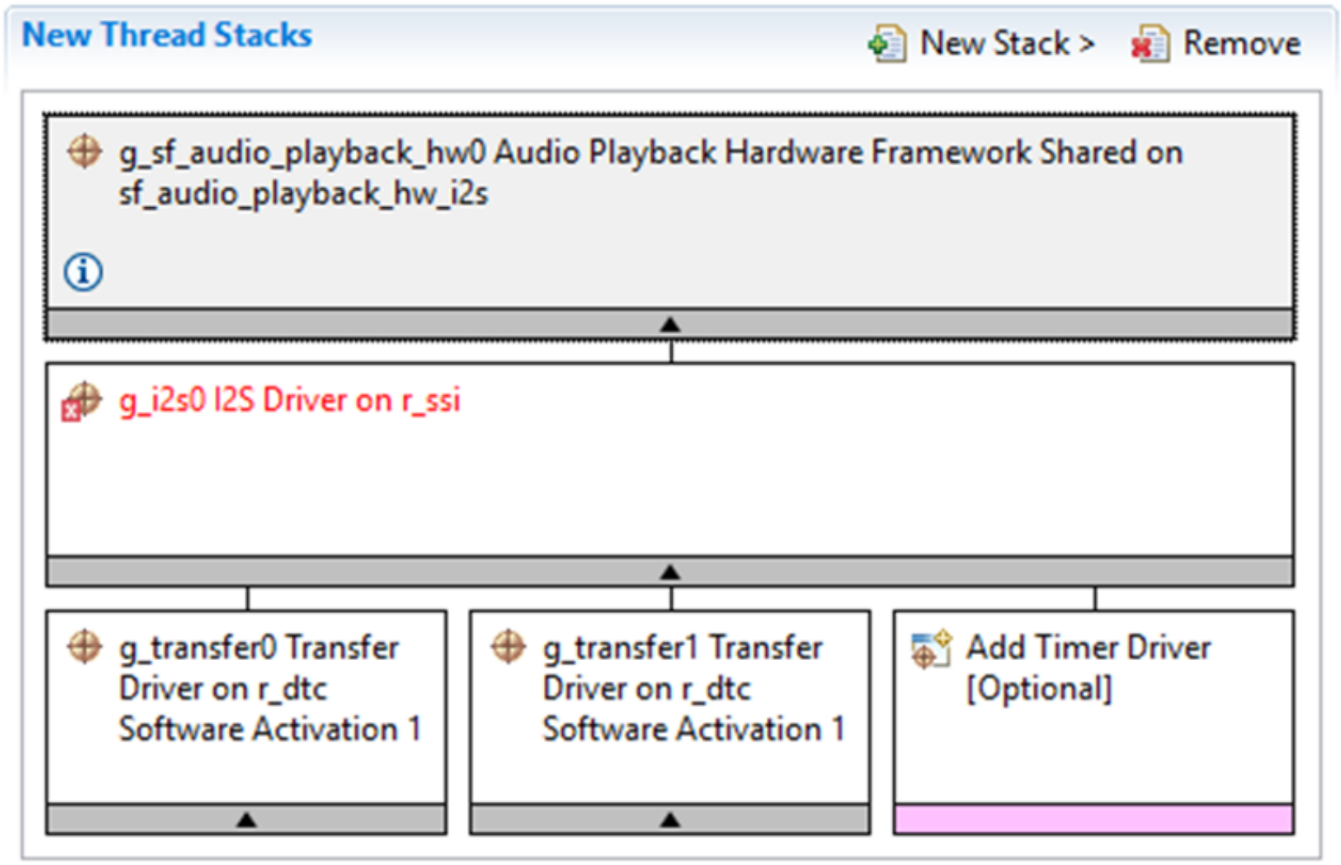


Figure 119: Audio Playback I2S Framework Module Stack

4.1.4.5 Configuring the Audio Playback I2S Framework Module

The Audio Playback I2S Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Audio Playback I2S Framework Module on sf_audio_playback_hw_i2s

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_sf_audio_playback_hw0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, it might be useful to select the DAC or I2S Channel based on the target hardware implementation. The configurable properties for the lower level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Audio Playback I2S Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits).
Sampling Frequecy (Hertz)	44100	Sampling frequency of audio data.
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).

WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Transmit interrupt priority selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Receive interrupt priority selection.
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Idle/error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.

Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.

Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.

Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Hertz	See Timer Period Calculation.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.

Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.
Period Unit	Hertz	Period unit selection.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.

Auto Start	FALSE	Auto start selection.
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection.
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection.
Callback	NULL	Callback selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Audio Playback I2S Framework Module Clock Configuration

The Audio Playback I2S Framework hardware modules use the peripheral clocks available in the clock configuration window.

Audio Playback I2S Framework Module Pin Configuration

The SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following tables illustrate the method for selecting the pins within the SSP configuration window and shows an example selection for the associated pins.

Pin Selection Sequence for the Audio Playback I2S Framework Module

Resource	ISDE Tab	Pin selection Sequence
I2S	Pins	Select **Peripherals** > **Connectivity:SSI** > **SSI0/SSI1**.

Note

The selection sequence assumes the ADC0/ADC1 or the SSI0/SSI1 is the desired hardware target of the driver.

Pin Configuration Settings for the I2S Driver on r_ssi

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port.
Operation Mode	Enabled, Custom, Disabled	Operation selection.
SSISCK	None, P204 Default: None	SSI Serial clock.
SSIWS	None, P205 Default: None	SSI Stereo pin selection.
SSIDATA	None, P206 Default: None	SSI Data pin selection.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.4.6 Using the Audio Playback I2S Framework Module in an Application

The typical steps in using the Audio Playback I2S Framework module in an application are:

1. Initialize the Audio Playback I2S Framework using the sf_audio_playback_hw_api_t::open API.

2. Start the low level hardware of the Audio Playback I2S framework using the sf_audio_playback_hw_api_t::start API.

3. Play the PCM audio samples using the sf_audio_playback_hw_api_t::play API.

4. PCM audio samples supported by the hardware is provided by the sf_audio_playback_hw_api_t::dataTypeGet API.

5. Stop the low level hardware of the Audio Playback I2S Framework using sf_audio_playback_hw_api_t::stop API.

6. Close the Audio Playback I2S framework using the sf_audio_playback_hw_api_t::close API.

These common steps are illustrated in a typical operational flow in the following figure:

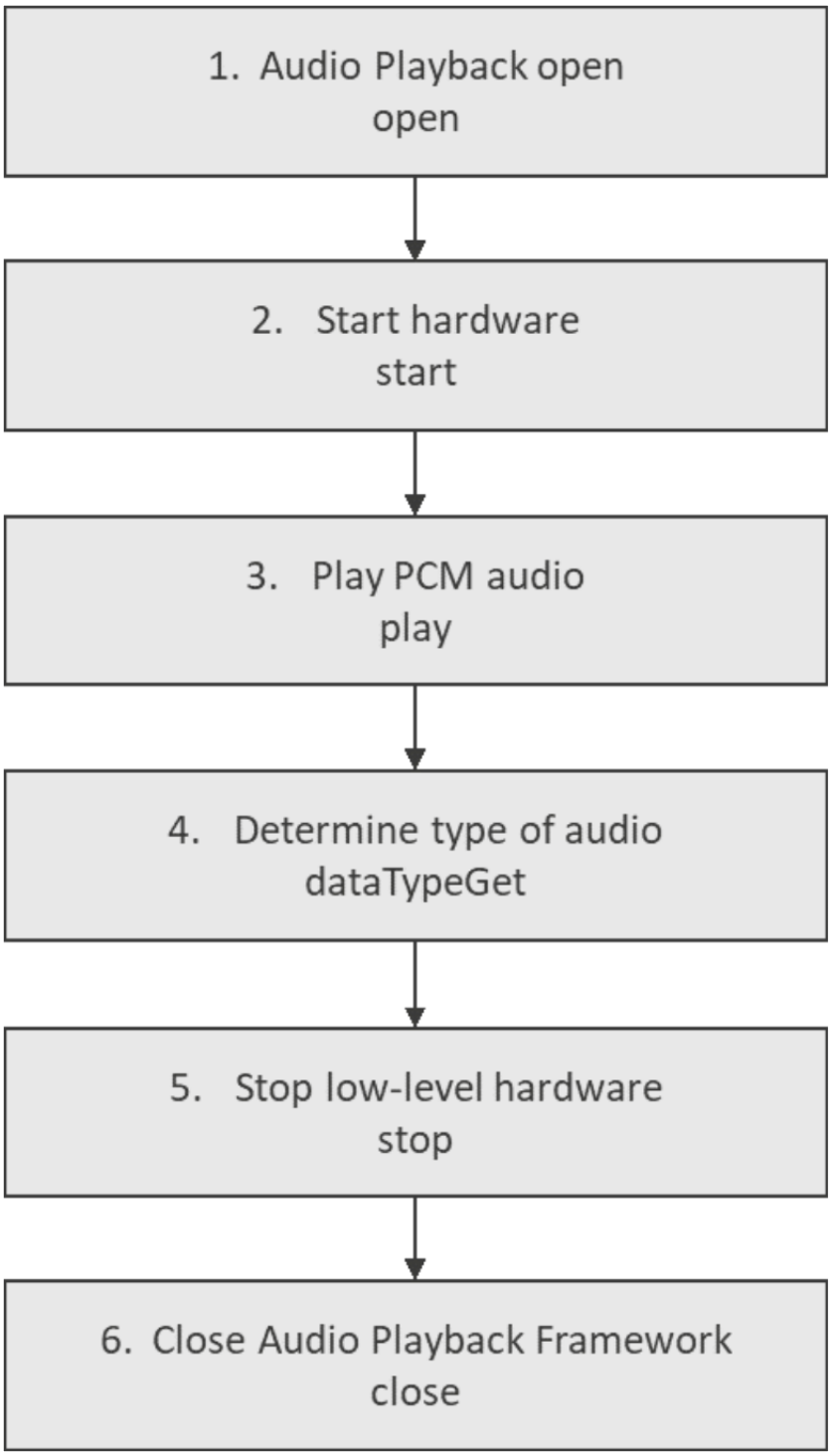


Figure 120: Flow Diagram of a Typical Audio Playback I2S Framework Module Application

4.1.5 Audio Record ADC Framework

4.1.5.1 Audio Record ADC Framework Module Introduction

The Audio Record ADC Framework module provides a high-level API for audio recording applications and uses the sf_adc_periodic and its lower layer ADC, GPT and DTC peripherals on the Synergy MCU. A user-defined callback can be created to indicate that the sample count has been completed.

Audio Record ADC Framework Module Features

- Currently supports 12-bit ADCs (supports 8, 10, and 12 bits) and 14-bit ADCs (supports 14 or 12-bit PCM data)
- Uses ADC Periodic Framework to simplify configuration and integration
- Uses a ThreadX object (for example, mutex) to protect hardware from improper access
- APIs for high-level functions simplify coding:
    - sf_audio_record_api_t::open, sf_audio_record_api_t::start
    - sf_audio_record_api_t::stop, sf_audio_record_api_t::infoGet
    - sf_audio_record_api_t::close

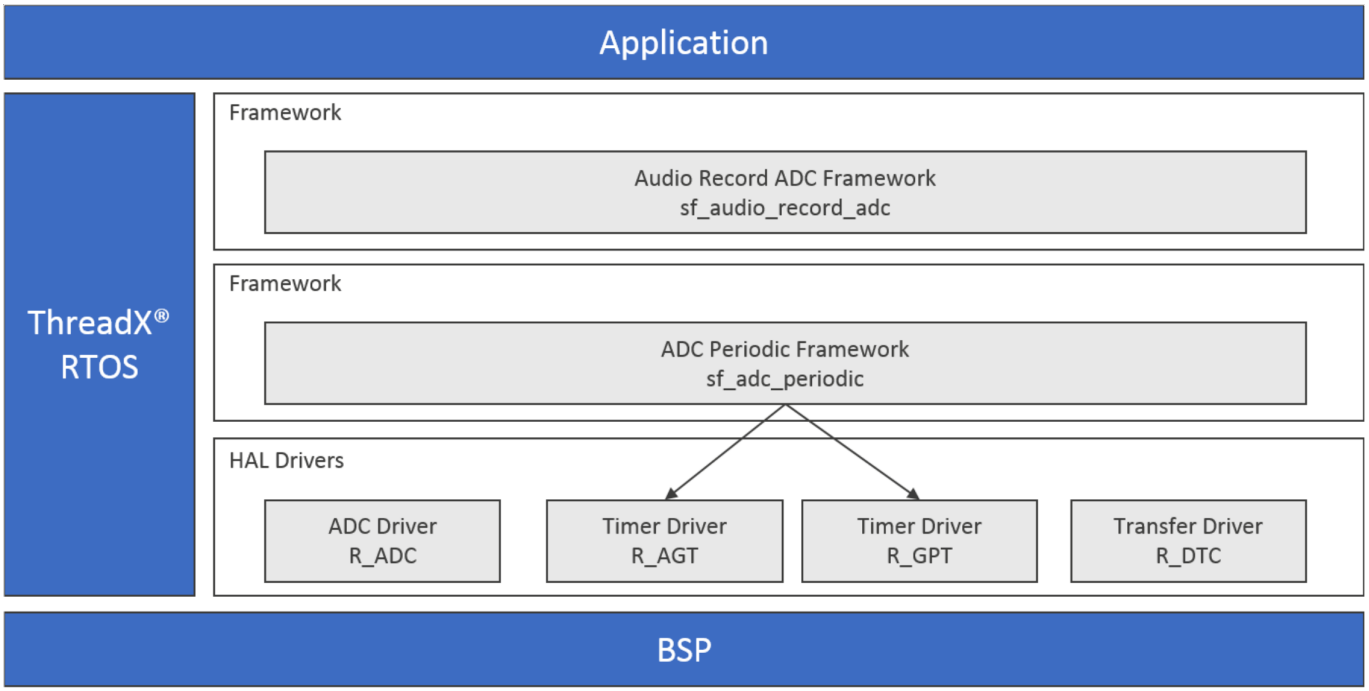


Figure 121: Audio Record ADC Framework Module Block Diagram

4.1.5.2 Audio Record ADC Framework Module APIs Overview

The Audio Record ADC Framework module defines APIs for opening, closing, starting and stopping the record process. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Record ADC Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_audio_record_adc.p_api->open(g_sf_audio_record_adc.p_ctrl, g_sf_audio_record_adc.p_cfg); Initialize the module.

start	g_sf_audio_record_adc.p_api->start(g_sf_audio_record_adc.p_ctrl); Start audio recording.
stop	g_sf_audio_record_adc.p_api->stop(g_sf_audio_record_adc.p_ctrl); Stop audio recording.
infoGet	g_sf_audio_record_adc.p_api->infoGet(g_sf_audio_record_adc.p_api.p_ctrl; Get the channel information (mono or Stereo).
close	g_sf_audio_record_adc.p_api->close(g_sf_audio_record_adc.p_ctrl); Close the module.
versionGet	g_sf_audio_record_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUME	Parameter has invalid value.
SSP_ERR_IN_USE	The adc periodic framework mutex may be unavailable for the unit requested. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_api->close is NULL).

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.5.3 Audio Record ADC Framework Module Operational Overview

The Audio Record ADC Framework Module samples audio analog data using the ADC Periodic Framework and the data samples captured are stored in the user buffer. The data is made available for further processing as needed by the application. The Audio Record ADC Framework has a configuration parameter that is initialized during the framework initialization, which also initializes

the underlying ADC periodic framework for data capture.

The captured data is stored in a user defined buffer and this is done in the callback function as illustrated below:

Assuming the name of the callback has been configured to be sf_audio_record_user_callback:

```
uint16_t * audio_record_buffer;
void sf_audio_record_user_callback (sf_audio_record_callback_args_t *p_args)
{
audio_record_buffer = ((uint16_t *)g_sf_audio_record_adc.p_cfg->
p_capture_data_buffer + (p_args->buffer_index/2)); }
```

Audio Record ADC Framework Module Important Operational Notes and Limitations

Audio Record ADC Framework Module Operational Notes

- The Audio Record ADC Framework Module configuration data can specify the length of the data buffer, data width, sampling rate and the number of sampling iterations.

Audio Record ADC Framework Module Limitations

- Currently, the Audio Record ADC only supports the ADC Periodic Framework as the lower level; recording via the I2S is not supported.
- The framework currently supports recording 8 bit or 12 bit PCM data.
- Currently, the Audio Record ADC only supports mono channel.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.5.4 Including the Audio Record ADC Framework Module in an Application

This section describes how to include the Audio Record ADC Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Record ADC Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the ADC Audio Record ADC Framework is g_adc_record_adc0. This name can be changed in the associated Properties window.)

Audio Record ADC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_audio_record_adc0 Audio Record ADC Framework on sf_audio_record_adc	Threads	**New Stack> Driver> Audio> Audio Record ADC Framework on sf_audio_record_adc**

When the Audio Record ADC Framework on sf_audio_record_adc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

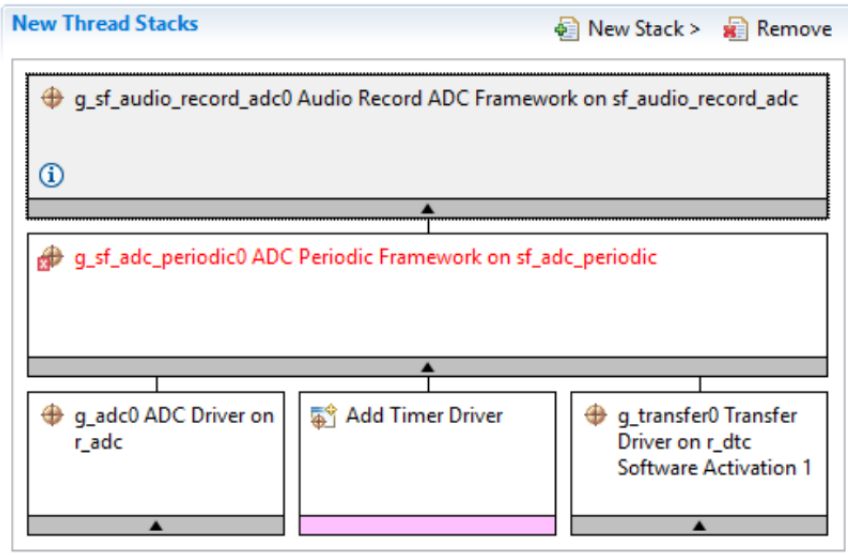


Figure 122: Audio Record ADC Framework Module Stack

4.1.5.5 Configuring the Audio Record ADC Framework Module

The Audio Record ADC Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Audio Record ADC Framework Module on sf_audio_record_adc

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_audio_record_adc0	Module name.
Name of the data-buffer to store samples	p_capture_data_buffer	Name of the 16-bit data buffer to store samples.
Length of the data-buffer	2048	Length of the buffer to which data is to be stored.
Audio Record Data Size	8-Bit, 16-Bit Default: 8-Bit	The data width of captured data 8 bit or 16 bit.
Sampling Rate in HZ	8000	Sampling rate to be used to capture data.
Number of sampling iterations	256	Samples to be captured.
Callback	g_audio_record_framework_user_callback	Callback to user after capturing the sample count.
Name of generated initialization function	sf_audio_record_adc_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Audio Record ADC Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the ADC Periodic Framework on sf_adc_periodic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_adc_periodic0	Module name.
Name of the data-buffer to store samples	g_user_buffer	Name of the 16-bit data buffer to store samples.
Length of the data-buffer	2048	Length of the buffer to which data is to be stored.

Number of sampling iterations	256	Priority of ADC Periodic Framework internal thread.
Callback	NULL	User function that will be called once "sample_counts" number of data has been buffered.
Name of generated initialization function	sf_adc_periodic_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected code for parameter checking is included in the build.
Name	g_adc0	Module name.
Unit	0, 1 (S7G2 Only) Default: 0	Specify the ADC Unit to be used. The S7G2 has two units; 0 and 1.
Resolution	14-Bit (S3A7/S124 Only), 12-Bit, 10-Bit (S7G2) Default: 8-Bit (S7G2 Only)	Specify the conversion resolution for this unit.
Alignment	Right, Left Default: Right	Specify the conversion result alignment.
Clear after read	Off, On Default: On	Specify if the result register must be automatically cleared after the conversion result is read. Note: If this is enabled, then watching the result register using a debugger always results in a 0.
Mode	Single Scan	The ADC Framework preconfigures and locks this field.

Channels 0-6	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 7-10 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 11-15 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 16-20	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 21 (Unit 0 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

Channel 22 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 23-27 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Temperature Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Temperature sensor use selection for Channel Scan Mask.
Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Voltage sensor use selection for Channel Scan Mask.
Normal/Group A Trigger	ELC Event	The ADC Framework preconfigures and locks this field.
Group B Trigger (Valid Only in Group Scan Mode)	ELC Event (The only valid trigger for either group in Group Scan Mode)	The ADC Framework preconfigures and locks this field.
Group Priority (Valid only in Group Scan Mode)	Group A cannot interrupt Group B, Group A can interrupt Group B; Group B scan restarts at next trigger, Group A can interrupt Group B; Group B scan restarts immediately, Group A can interrupt Group B; Group B scan restarts immediately and scans continuously Default: Group A cannot interrupt Group B	Do not use with ADC Framework since the mode is locked to Single Scan Mode.

Add/Average Count	Disabled, Add two samples, Add three samples, Add four samples, Add sixteen samples, Average two samples, Average four samples Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask adc_channel_cfg_t::add_mask.
Channels 0-27	Disabled, Enabled Default: Disabled	This field is valid only if adc_cfg_t::add_average_count is enabled. This field determines what channels results are to be averaged or summed.
Temperature Sensor	Disabled, Enabled Default: Disabled	Temperature sensor use selection for Addition/Averaging Mask.
Voltage Sensor	Disabled, Enabled Default: Disabled	Voltage sensor use selection for Addition/Averaging Mask.
Channels 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in adc_channel_cfg_t::sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.
Sample Hold States (Applies only to the 3 channels selected above)	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if adc_channel_cfg_t::sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. Note: Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.
Callback	NULL	The ADC Framework uses the callback internally.
Scan End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Interrupt Priority selection.

Scan End Group B Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Group B Interrupt Priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Microseconds	See Timer Period Calculation.
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.

AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: True	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Underflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.

Channel	0	The ADC Framework preconfigures and locks this field based on channel selected in the ADC Framework.
Mode	Periodic	The ADC Framework preconfigures and locks this field.
Period Value	10	Configure timer period to trigger ADC scans.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Milliseconds	Configure units of the timer period set above.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	False	The ADC Framework preconfigures and locks this field.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	The ADC Framework preconfigures and locks this field.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Block	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	1	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

Audio Record ADC Framework Module Clock Configuration

The ADC peripheral module uses the PCLKC as its clock source.

Audio Record ADC Framework Module Pin Configuration

The ADC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. ADC pins must be configured as analog pins. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the Audio Record ADC Framework Module on sf_audio_record_adc

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select **Peripherals** > **Analog:ADC** > **ADC0**

Note

The selection sequence assumes KINT0 is the desired hardware target for the driver.

Pin Configuration Settings for the Audio Record ADC Framework Module on sf_audio_record_adc

Property	Value	Description
Operation Mode	Disabled, Custom Default: Custom	Select operating mode for ADC.
ADTRG	None, P407, P102 Default: None)	ADTRG pin.
AN00-19	None, Pnnn, Pmmm Default: None	Analog input pins.
PGAVSS0	None, P003 Default: None	PGAVSS pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.1.5.6 Using the Audio Record ADC Framework Module in an Application

The steps in using the Audio Record ADC Framework module on sf_audio_record_adc in a typical application are:

1. Open the module using the sf_audio_record_api_t::open API.
2. Start the recording using the sf_audio_record_api_t::start API.
3. Store data in a user buffer with the callback.

4. Operate on data as needed.
5. Close the module using the sf_audio_record_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

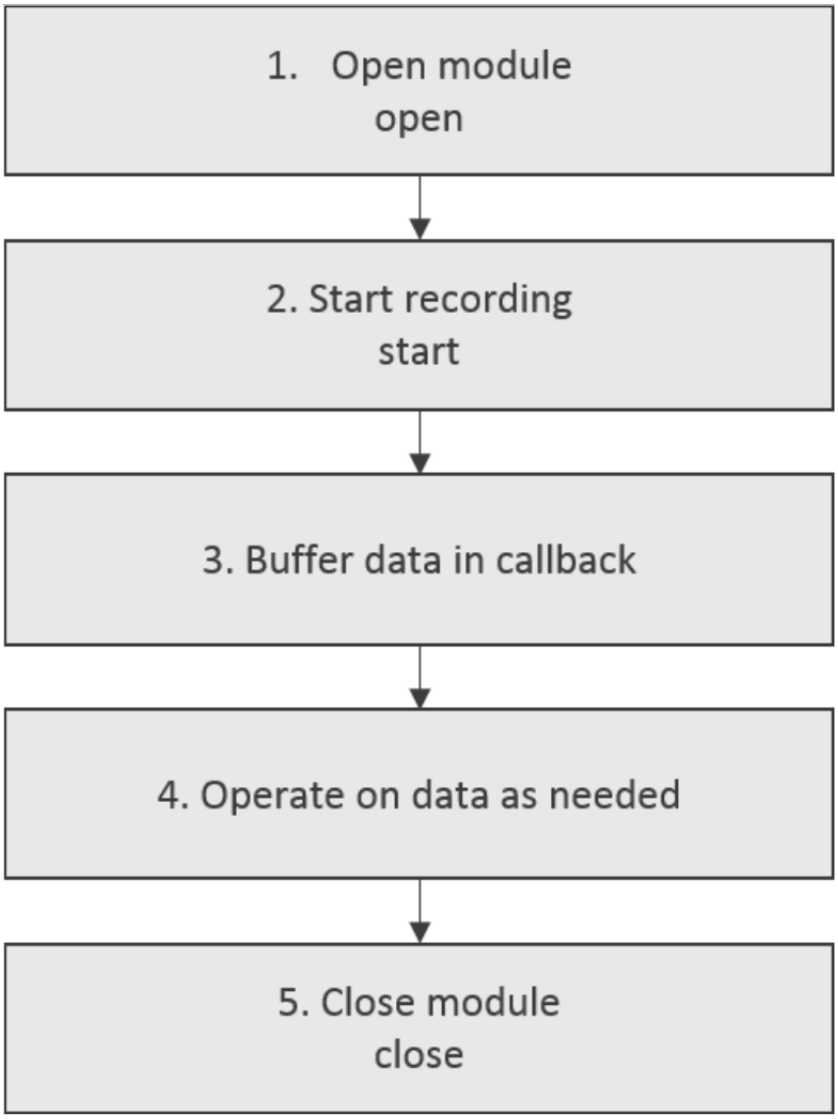


Figure 123: Flow Diagram of a Typical Audio Record ADC Framework Module Application

4.1.6 Audio Record I2S Framework

4.1.6.1 Audio Record I2S Framework Introduction

The Audio Record I2S Framework module provides a high-level API for audio recording applications and uses the I2S interface. The Audio Record I2S Framework module uses the SSI, GPT and DTC peripherals on the Synergy MCU. A user-defined callback can be created to indicate that new samples are stored in the user buffer.

Audio Record I2S Framework Module Features

- Thread safe
- Records data in 8 or 16-bit PCM
- Periodic callback function when new samples are available
- Configurable number of samples (sample count) per callback

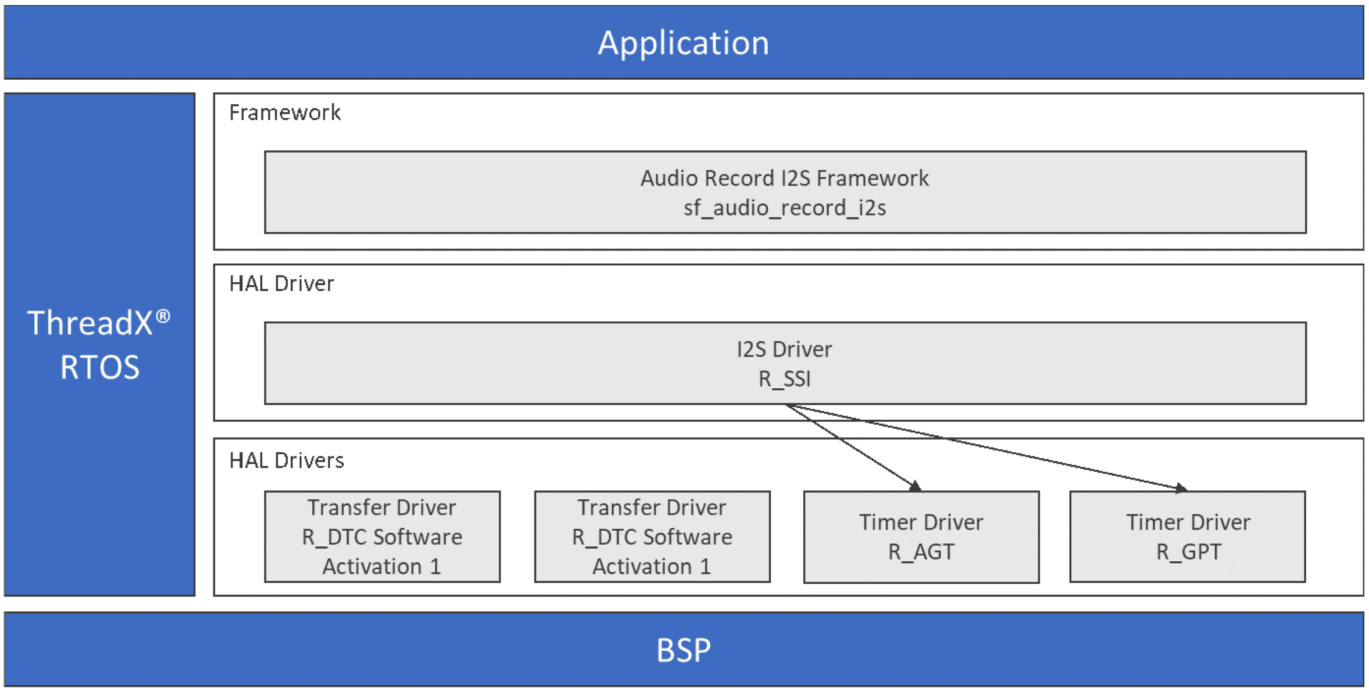


Figure 124: Audio Record I2S Framework Module Block Diagram

4.1.6.2 Audio Record I2S Framework Module APIs Overview

The Audio Record I2S Framework module defines APIs to open, start, stop and close the recording module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Record I2S Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_audio_record_i2s0.p_api->open (g_sf_audio_record_i2s0.p_ctrl, g_sf_audio_record_i2s0.p_cfg); Initializes audio recording framework.
start	g_sf_audio_record_i2s0.p_api->start (g_sf_audio_record_i2s0.p_ctrl); Starts audio recording.
stop	g_ sf_audio_record_i2s0.p_api->stop (g_sf_audio_record_i2s0.p_ctrl); Stops audio recording.

infoGet	g_sf_audio_record_i2s0.p_api->infoGet (g_sf_audio_record_i2s0.p_ctrl, p_info); Gets channel information (Mono/Stereo).
close	g_sf_audio_record_i2s0.p_api->close (g_sf_audio_record_i2s0.p_ctrl); Releases channel mutex and closes channel at HAL layer.
versionGet	g_ sf_audio_record_i2s0.p_api->versionGet(&version); Gets version and stores it in provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.
SSP_ERR_UNSUPPORTED	Command not supported.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.6.3 Audio Record I2S Framework Module Operational Overview

The Audio Record I2S Framework Module uses the I2S HAL layer as the underlying interface for the audio data transfer and the data captured is stored in the user buffer. The data is made available for further processing as needed by the application.

The captured data is stored in a user defined buffer and this is done in the callback function as illustrated below:

Assuming the name of the callback has been configured to be sf_audio_record_user_callback.

```
uint16_t * audio_buffer;

void audio_record_user_callback (sf_audio_record_callback_args_t * p_args)
```

```
{
    audio_buffer = ((uint16_t *)sf_audio_record_i2s.p_cfg->p_capture_data_buffer
                   + (p_args->buffer_index));
}
```

Audio Record I2S Framework Module Important Operational Notes and Limitations

Audio Record I2S Framework Module Operational Notes

The Audio Record I2S Framework Module configuration data can specify the name of the data buffer, length of the data buffer, data size (8-bit or 16-bit samples), sampling iterations and the name of the callback.

Audio Record I2S Framework Module Limitations

- The framework currently supports recording 8-bit or 16-bit PCM data.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.6.4 Including the Audio Record I2S Framework Module in an Application

This section describes how to include the Audio Record I2S Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Record I2S Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Audio Record I2S Framework module is g_sf_audio_record_i2s0. This name can be changed in the associated Properties window.)

Audio Record I2S Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_record_i2s0 Audio Record I2S Framework on sf_audio_record_i2s	Threads	**New Stack> Framework> Audio> Audio Record I2S Framework on sf_audio_record_i2s**

When the Audio Record I2S Framework module on sf_audio_record_i2s is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink

banded modules brings up the New icon and displays possible choices.

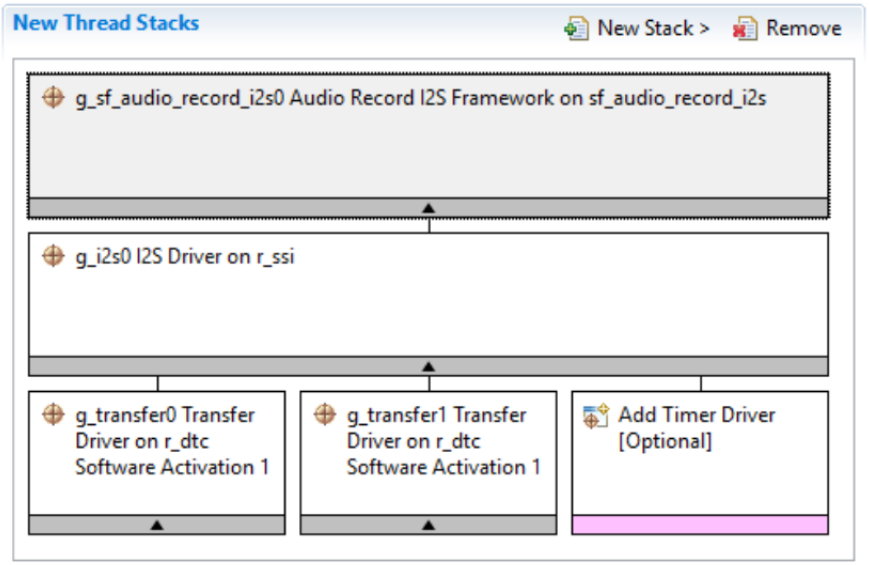


Figure 125: Audio Record I2S Framework Module Stack

4.1.6.5 Configuring the Audio Record I2S Framework Module

The Audio Record I2S Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Audio Record I2S Framework Module on sf_audio_record_i2s

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_audio_record_i2s0	Module name.
Name of the data-buffer to store samples	p_capture_data_buffer	Data-buffer name.
Length of the data buffer	2048	Length of the data buffer.

Audio Record Data Size	8-Bit, 16-Bit Default: 16-Bit	Audio record data size selection.
Number of sampling iterations	256	Number of sampling iterations.
Callback	g_audio_record_framework_user_callback	Callback name.
Name of generated initialization function	sf_audio_record_i2s_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Audio Record I2S Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits).
Sampling Frequency (Hertz)	44100	Sampling frequency of audio data.

Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Audio Clock Frequency (Hertz)	External, GTIOC1A Default: External	Select External for external signal to AUDIO_CLK input pin or GTIOCA1.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Transmit interrupt priority selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Receive interrupt priority selection.
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Idle/error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.
Period Unit	Hertz	Period unit selection.
Auto Start	FALSE	Auto start selection.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	Count source selection.
AGTO Output Enabled	True, False Default: False	AGTO output selection.
AGTIO Output Enabled	True, False Default: False	AGTIO output selection.
Output Inverted	True, False Default: False	Output inverted selection.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	Callback selection.
Underflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.
Period Unit	Hertz	Period unit selection.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	FALSE	Auto start selection.
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection.
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection.
Callback	NULL	Callback selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Audio Record I2S Framework Module Clock Configuration

The Audio Record I2S Framework module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Audio Record I2S Framework Module Pin Configuration

To use the Audio Record I2S Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the Audio Record I2S Framework Module

Resource	ISDE Tab	Pin selection Sequence
SSI	Pins	Select **Peripherals**> **Connectivity:SSI**> **SSI/SSI0/SSI1**

4.1.6.6 Using the Audio Record I2S Framework Module in an Application

The typical steps in using the Audio Record I2S Framework module in an application are:

1. Initialize the module using the sf_audio_record_api_t::open API.
2. Start the recording using the sf_audio_record_api_t::start API.
3. Operate on data from the periodic callback function as needed.
4. Close the module using the sf_audio_record_api_t::close API.

These common steps are illustrated in a typical operational flow in the following figure:

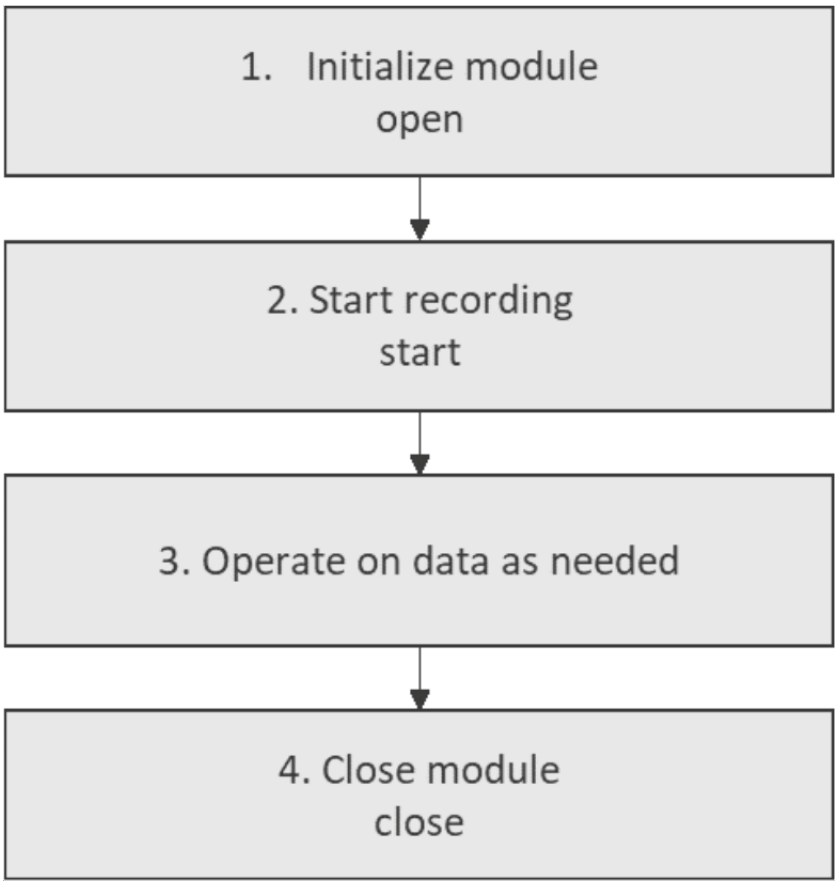


Figure 126: Flow Diagram of a Typical Audio Record I2S Framework Module Application

4.1.7 Block Media Framework on sf_block_media_lx_nor

4.1.7.1 Block Media Framework Module Introduction

The Block Media Framework on sf_block_media_lx_nor module provides a high-level API for interfacing with QSPI NOR flash memory devices. It provides API functions for reading, writing and controlling the QSPI NOR Flash memory. The framework includes the Azure RTOS wear leveling component LevelX NOR. The LevelX functions to support wear leveling are automatic and transparent to the developer. File system accesses, using the Azure RTOS FileX system, are also supported, making it easy to implement file system based applications.

Block Media Framework Module Features

- Supports Block Media Framework interface for NOR flash memory device.
- Supports file system access on NOR flash memory.
- Supports LevelX wear leveling functions transparently to the developer.

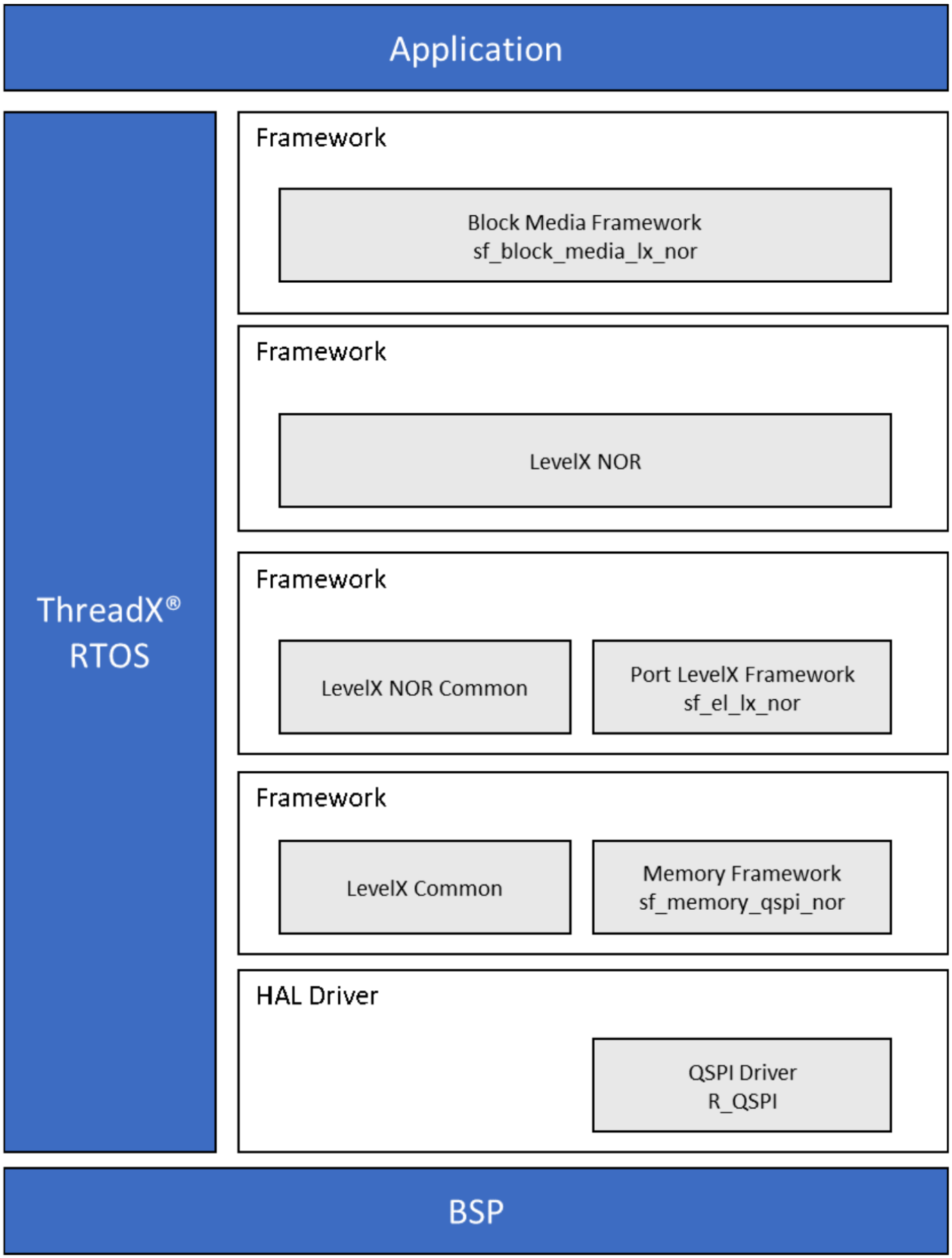


Figure 127: Block Media Framework Module Block Diagram

4.1.7.2 Block Media Framework Module APIs Overview

The Block Media Framework module defines API functions to open, read from, write to and close the module. A complete list of the available API functions, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_block_media_lx_nor0.p_api->open(g_sf_block_media_lx_nor0.p_ctrl, g_sf_block_media_lx_nor0.p_cfg); Open LevelX flash device for read/write and control. This function initializes the LevelX driver and hardware the first time it is called out of reset. The underlying flash needs to either be erased or already initialized with LevelX.
read	g_sf_block_media_lx_nor0.p_api->read(g_sf_block_media_lx_nor0.p_ctrl, p_dest, start_sector, sector_count); Read data from flash using LevelX.
write	g_sf_block_media_lx_nor0.p_api->write(g_sf_block_media_lx_nor0.p_ctrl, p_src, start_sector, sector_count); Write data to flash using LevelX.
ioctl	g_sf_block_media_lx_nor0.p_api->ioctl(g_sf_block_media_lx_nor0.p_ctrl, command, p_data); Send control commands to Block Media LevelX NOR driver.
close	g_sf_block_media_lx_nor0.p_api->close(g_sf_block_media_lx_nor0.p_ctrl); Close an open Block Media LevelX NOR driver.
versionGet	g_sf_block_media_lx_nor0.p_api->versionGet(&version); Return the version of the firmware and API using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	LevelX flash is available and is now open for read, write and control access.

SSP_ERR_ASSERTION	p_ctrl, p_cfg or an input pointer is NULL.
SSP_ERR_ALREADY_OPEN	The block media LevelX NOR instance has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the instance.
SSP_ERR_MEDIA_OPEN_FAILED	LevelX NOR or the underlying flash failed to open. The underlying flash needs to either be erased or already initialized with LevelX.
SSP_ERR_NOT_OPEN	The block media is not open.
SSP_ERR_READ_FAILED	Data read failed.
SSP_ERR_WRITE_FAILED	Data write failed.
SSP_ERR_UNSUPPORTED	This module does not support requested command.
SSP_ERR_SECTOR_RELEASE_FAILED	Sector release command failed.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.7.3 Block Media Framework Module Operational Overview

The Block Media Framework on sf_block_media_lx_nor framework module provides a high-level API for interfacing with QSPI NOR flash memory devices while supporting wear leveling and file system operations. Wear leveling is implemented, transparently to the developer, using the Azure RTOS LevelX component within the SSP. File system support is implemented using the Azure RTOS FileX component, integrated with the SSP. These components work together to make it easy to support embedded applications that require QSPI NOR Flash file system operations.

The Block Media Framework provides API functions for reading, writing and controlling the QSPI NOR Flash memory. In addition to these functions, all FileX related API functions become available once the Block Media Framework module is successfully opened and initialized. Refer to the FileX User's Manual for a complete description of these functions.

The Block Media Framework module uses a standard interface that is common to other SSP media modules. For example, the modules that support SDMMC, SPI Flash and SDRAM/RAM memories use the same API calls, so the programming interface remains the same for any media driver. These modules can be interchanged with one another easily. Device adaptation drivers, such as r_qspi, are accessed through the Memory Framework interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through memory interface function calls are generally device specific as well.

Block Media Framework Module Important Operational Notes and Limitations

Block Media Framework Module Operational Notes

The media must be erased and formatted before using sf_block_media_lx_nor for creating, writing and reading files.

Block Media Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.7.4 Including the Block Media Framework Module in an Application

This section describes how to include the Block Media Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Block Media Framework is g_sf_block_media_lx_nor. This name can be changed in the associated Properties window.)

Block Media Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_block_media_lx_nor0 Block Media Framework on sf_block_media_lx_nor	Threads	**New Stack> Framework> File System> Block Media Framework on sf_block_media_lx_nor**

When the Block Media Framework on sf_block_media_lx_nor is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

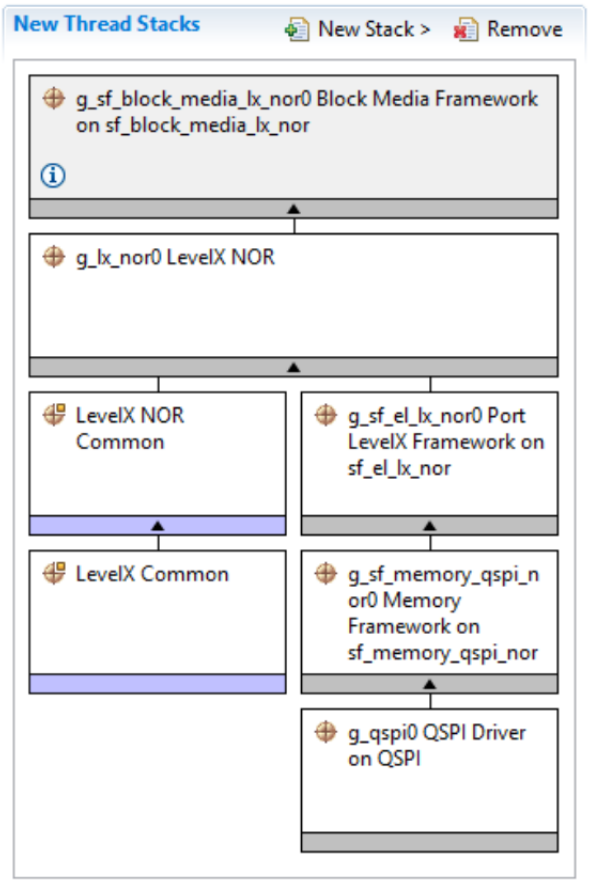


Figure 128: Block Media Framework Module Stack

4.1.7.5 Configuring the Block Media Framework Module

The Block Media Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Block Media Framework Module on sf_block_media_lx_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.

Name	g_sf_block_media_lx_nor0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Block Media Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the LevelX NOR Common Instance

ISDE Property	Value	Description
Name	g_lx_nor0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the LevelX Common Instance

ISDE Property	Value	Description
Thread Safety	Enabled, Disabled Default: Disabled	If Enabled, this makes LevelX thread-safe by using a ThreadX mutex object throughout the API.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Port LevelX Framework Module on sf_el_lx_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_el_lx_nor0	Module name.
Event Callback	NULL	Name of the function to call when an event occurs.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Memory Framework on sf_memory_qspi_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_memory_qspi_nor0	Module name.
Write of Erase Timeout (in ticks)	30000	Timeout ticks for waiting on write or erase to complete.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_qspi0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Block Media Framework Module Clock Configuration

The Block Media framework module uses the QSPI peripheral which uses PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Block Media Framework Module Pin Configuration

To use the Block Media framework module, the port pins for the QSPI peripheral must be set as needed. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the Block Media Framework Module on sf_block_media_lx_nor

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select **Peripherals**> **Storage:QSPI QSPI0**

4.1.7.6 Using the Block Media Framework Module in an Application

The typical steps in using the Block Media Framework module in an application are:

1. Initialize the media using the FileX API function fx_system_initialize (sf_el_fx calls it automatically).

2. Initialize the LevelX NOR using the API function lx_nor_flash_initialize (g_common_init calls it automatically).

3. Format the media using the FileX API function fx_media_format (sf_el_fx calls it automatically if "Format media during initialization" property is set to Enabled). This Format media is optional if media has already been formatted. Anything on the media will erase while media format. (Optional)

4. Open the media using the FileX API function fx_media_open (sf_el_fx opens the media automatically).

5. Read the media as required using the sf_block_media_api_t::read API function (Block Media Framework) or one of the FileX API functions, for example, fx_media_read(), fx_file_read().

6. Write to the media as required using the sf_block_media_api_t::write API function (Block Media Framework) or one of the FileX API functions, for example, fx_media_write(), fx_file_write().

Note

1. After a successful fx_media_open call, all FileX APIs can be used (not just read and write).
2. If the media has been formatted by FileX without LevelX earlier, erase it using the qspi_api_t::erase API function before initializing the file media. The erase size should be not less than the size of the memory FileX uses. Users can disable the "Format media during initialization" property in the project configuration, erase the FileX media and then format it in the application.

These common steps are illustrated in a typical operational flow diagram in the following figure:

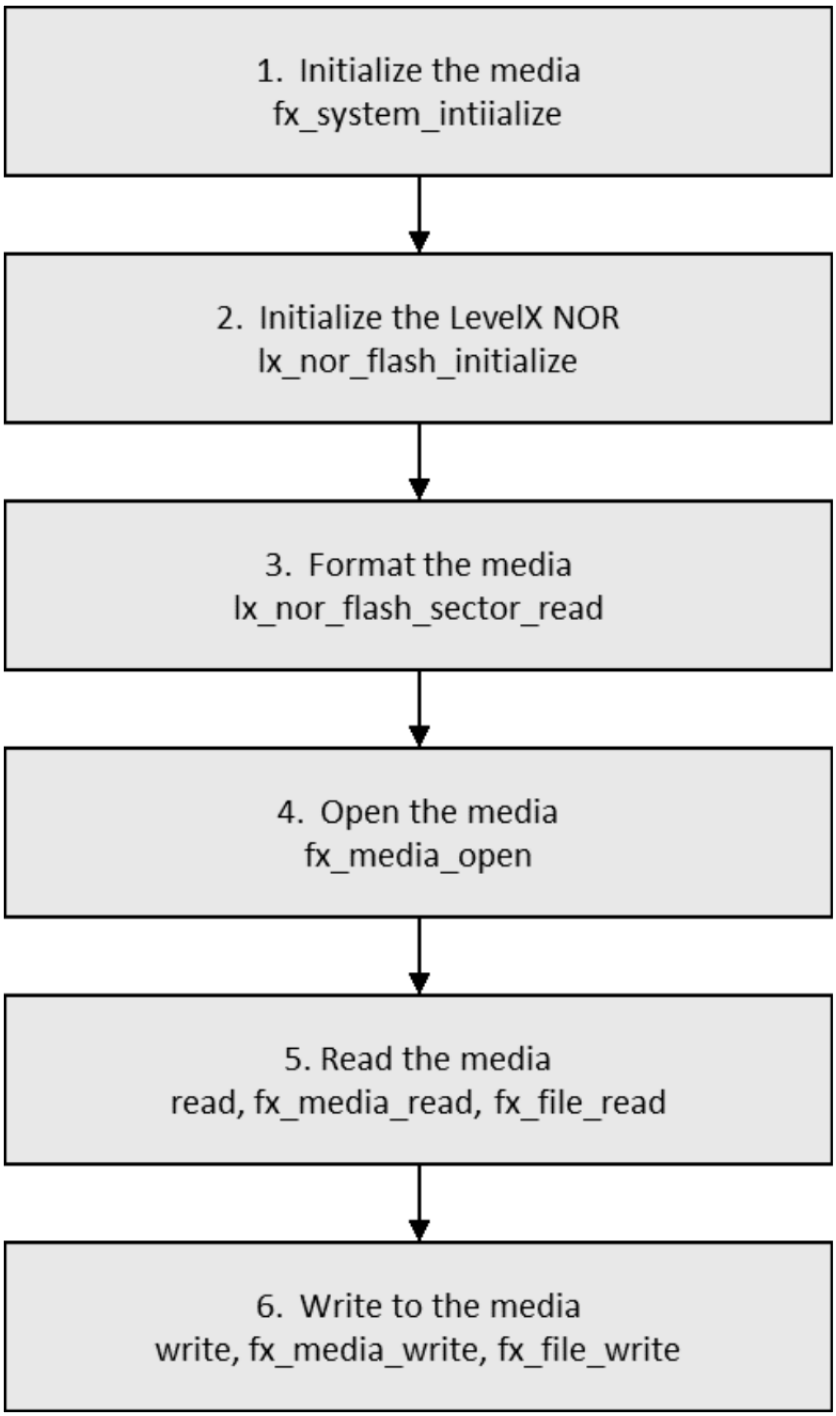


Figure 129: Flow Diagram of a Typical Block Media Framework Module Application

4.1.8 Block Media Framework on sf_block_media_qspi

4.1.8.1 Block Media QSPI Framework Module Introduction

The Block Media Framework Module can implement the QSPI channel for reading, writing and controlling the QSPI Flash memory peripheral through the r_qspi driver. The driver has all the functionality needed to interface with a file system through a block media interface.

Block Media QSPI Framework Module Features

- Supports QSPI channel interface for QSPI flash memory device.
- Support file system on QSPI flash memory.

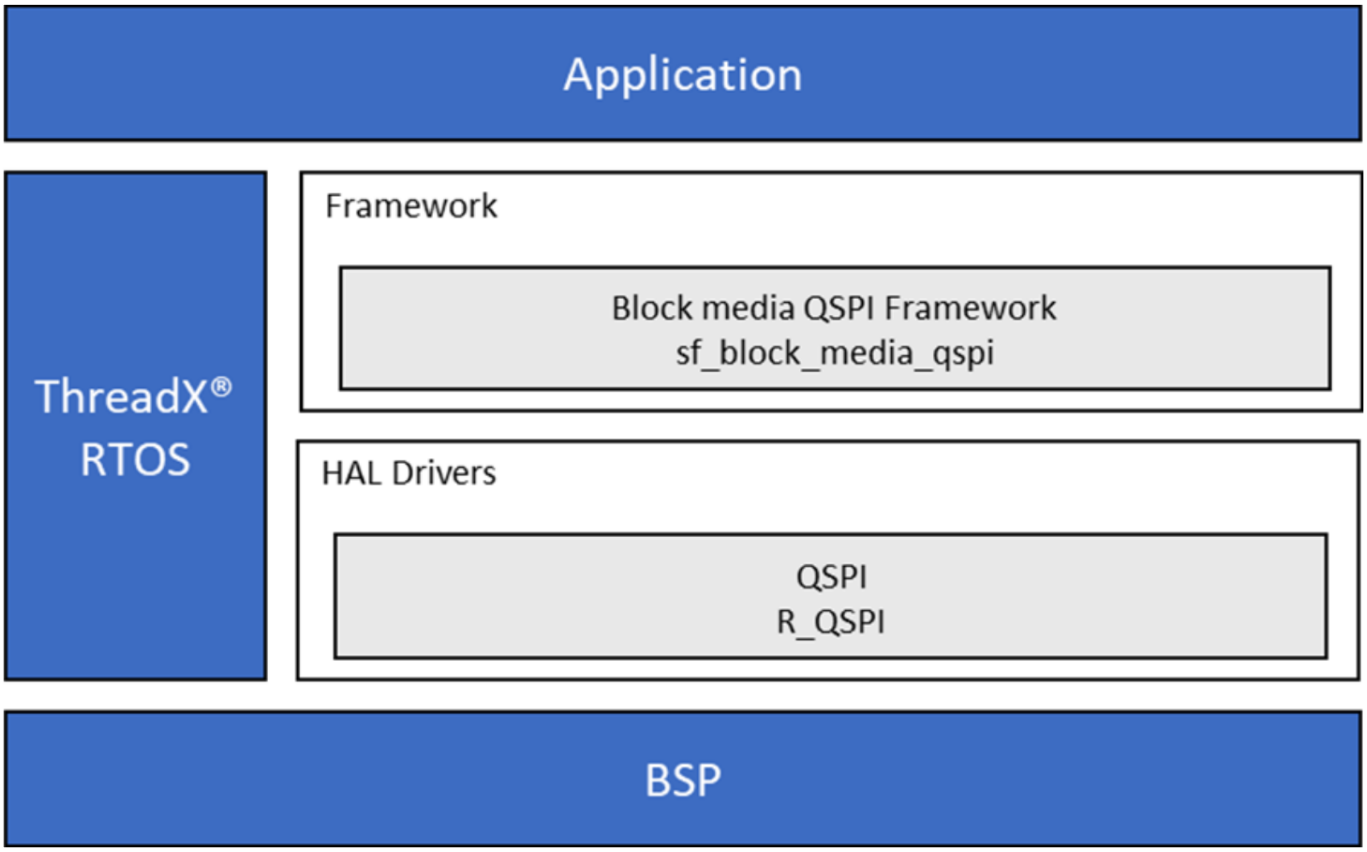


Figure 130: Block Media QSPI Framework Module Block Diagram

4.1.8.2 Block Media QSPI Framework Module APIs Overview

The Block Media QSPI Framework module defines APIs to open, control, read and write to the media. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media QSPI Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_block_media_qspi0.p_api->open (g_sf_block_media_qspi0.p_ctrl, g_sf_block_media_qspi0.p_cfg); Open a device channel for read/write and control.

read	g_sf_block_media_qspi0.p_api->read (g_sf_block_media_qspi0.p_ctrl, p_dest, start_block, block_count); Read data from a media channel.
write	g_sf_block_media_qspi0.p_api->write (g_sf_block_media_qspi0.p_ctrl, p_src, start_block, block_count); Write data to a media channel.
ioctl	g_sf_block_media_qspi0.p_api->ioctl (g_sf_block_media_qspi0.p_ctrl, command, p_data); Send control commands to and receives the status from the media port.
close	g_sf_block_media_qspi0.p_api->close (g_sf_block_media_qspi0.p_ctrl); Close the open media channel.
versionGet	g_sf_block_media_qspi0.p_api->versionGet(&version); Return the version of the driver using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.
SSP_ERR_UNSUPPORTED	Command not supported.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.8.3 Block Media QSPI Framework Module Operational Overview

The Block Media Framework Interface is simply an abstract interface using function pointers instead of direct function calls. Functions are called between FileX and the SSP block media drivers, such as the SDMMC, SPI Flash and SDRAM/RAM. The interface remains the same for any media driver so all

media drivers appear functionally identical at file I/O layer and can be interchanged with one another without changing code. Device adaptation drivers, such as sf_block_media_qspi, are accessed through the Block Media Framework Interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through block media function calls are generally device specific as well.

Block Media QSPI Framework Module Important Operational Notes and Limitations

Block Media QSPI Framework Module Operational Notes

- The media must be formatted at least once before you can begin creating, writing and reading files.

Block Media QSPI Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.8.4 Including the Block Media QSPI Framework Module in an Application

This section describes how to include the Block Media QSPI Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media QSPI Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Block Media QSPI Framework is g_sf_block_media_qspi0. This name can be changed in the associated Properties window.)

Block Media QSPI Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_block_media_qspi0Block Media Framework on sf_block_media_qspi	Threads	**New Stack> Framework> File System> Block Media Framework on sf_block_media_qspi**

When the Block Media QSPI Framework on sf_block_media_qspi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

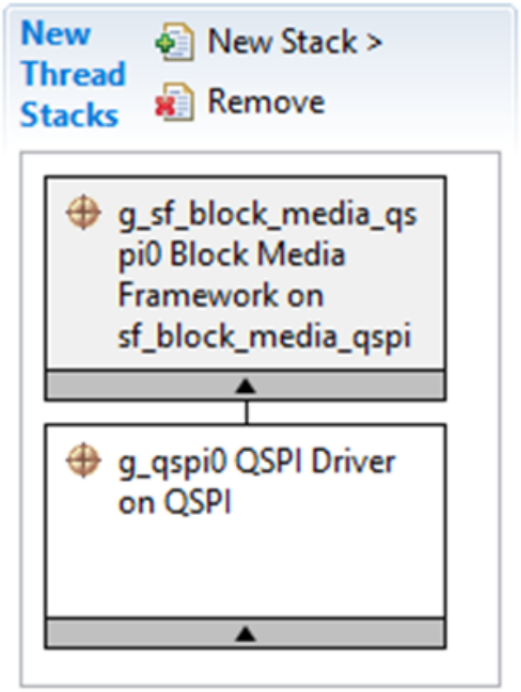


Figure 131: Block Media QSPI Framework Module Stack

4.1.8.5 Configuring the Block Media QSPI Framework Module

The Block Media QSPI Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Block Media QSPI Framework Module on sf_block_media_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_block_media_qspi	Module name.
Block size of media in bytes	4096	Block size selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Block Media QSPI Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_qspi0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Block Media QSPI Framework Module Clock Configuration

The Block Media QSPI Framework module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Block Media QSPI Framework Module Pin Configuration

To use the Block Media QSPI Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the Block Media QSPI Framework Module on sf_block_media_qspi

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select **Peripherals**> **Storage:QSPI**> **QSPI0**

4.1.8.6 Using the Block Media QSPI Framework Module in an Application

The steps in using the Block Media QSPI Framework module on sf_block_media_qspi in a typical application are:

1. Initialize the media using the FileX API fx_system_initialize (sf_el_fx calls it automatically).
2. Format the media using FileX API fx_media_format (sf_el_fx calls it automatically if "Format media during initialization" property is set to Enabled). This Format media is optional if media has already been formatted. Anything on the media will erase while media format.

(Optional)
3. Open the media using the FileX API fx_media_open (sf_el_fx opens the media automatically).
4. Read the media as required using the sf_block_media_api_t::read API (Block Media Framework) or one of the FileX APIs, for example, fx_media_read(), fx_file_read().
5. Write to the media as required using the sf_block_media_api_t::write API (Block Media Framework) or one of the FileX APIs, for example, fx_media_write(), fx_file_write().

These common steps are illustrated in a typical operational flow diagram in the following figure:

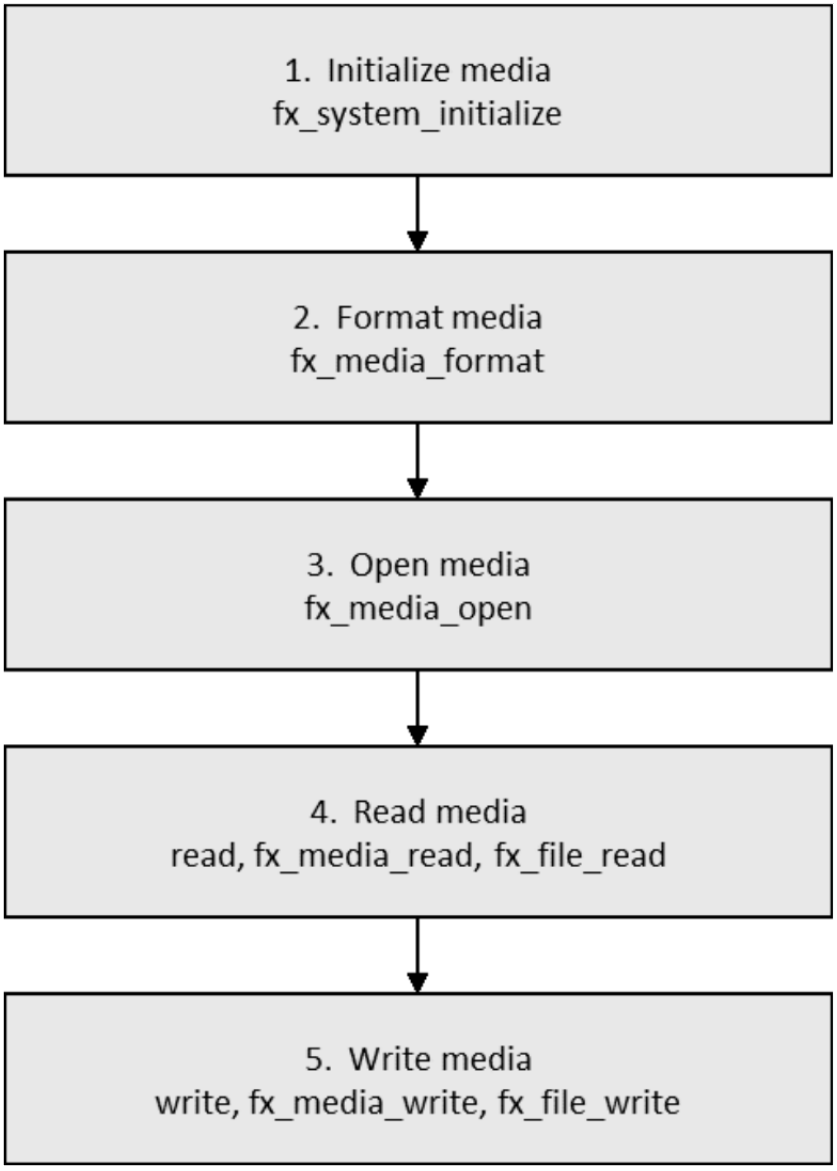


Figure 132: Flow Diagram of a Typical Block Media QSPI Framework Module Application

4.1.9 Block Media Framework on sf_block_media_ram

4.1.9.1 Block Media RAM Framework Module Introduction

The Block Media Framework Module can implement the file system on RAM for reading from, writing to and controlling the read/write region of the RAM memory. The framework has all the functionality needed to interface with a file system through a block media interface.

Block Media RAM Framework Module Features

- Enables FileX to be run on linear memory-mapped devices.
- Temporary and fast storage of data on RAM.

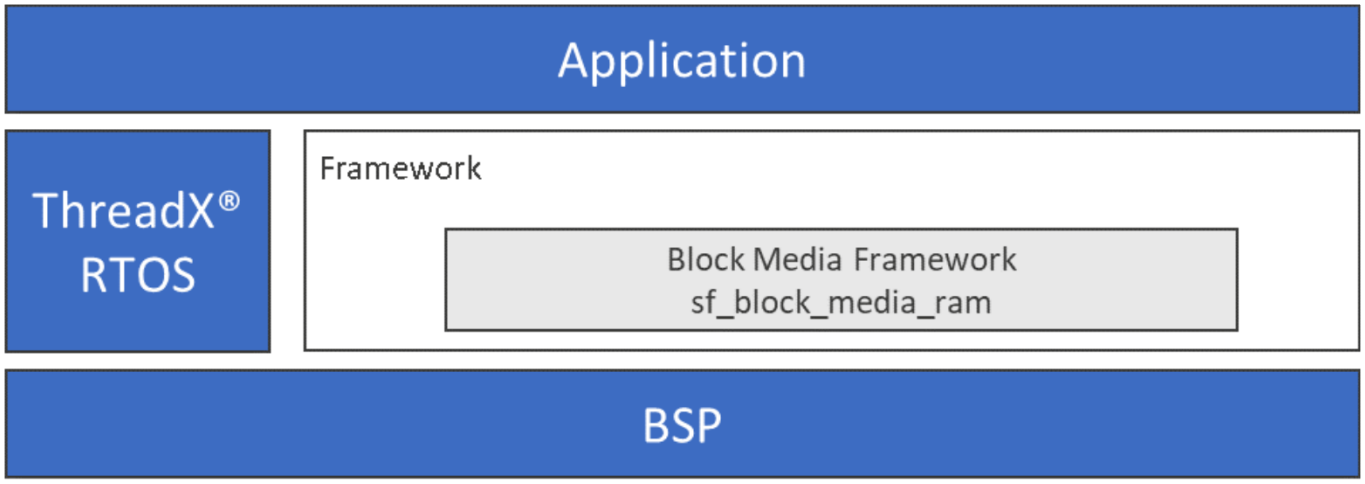


Figure 133: Block Media RAM Framework Module Block Diagram

4.1.9.2 Block Media RAM Framework Module APIs Overview

The Block Media RAM Framework module implements APIs to open, read from, write to and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media RAM Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_block_media_ram0.p_api->open (g_sf_block_media_ram0.p_ctrl, g_sf_block_media_ram0.p_cfg); Open device for read, write and control.
read	g_sf_block_media_ram0.p_api->read (g_sf_block_media_ram0.p_ctrl, p_dest, start_block, block_count); Read data from RAM buffer.
write	g_sf_block_media_ram0.p_api->write (g_sf_block_media_ram0.p_ctrl, p_source, start_block, block_count); Write data to RAM buffer.

ioctl	g_sf_block_media_ram0.p_api->ioctl (g_sf_block_media_ram0.p_ctrl, command, p_data); Send control commands to and receive status of RAM buffer.
close	g_sf_block_media_ram0.p_api->close (g_sf_block_media_ram0.p_ctrl); Close the framework.
versionGet	g_sf_block_media_ram0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_UNSUPPORTED	Command not supported.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	A parameter is NULL.
SSP_ERR_IN_USE	Framework is already open.
SSP_ERR_INVALID_BLOCKS	Invalid block passed.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.9.3 Block Media RAM Framework Module Operational Overview

The Block Media Framework Interface is simply an abstract interface using function pointers instead of direct function calls. Functions are called between the FileX and the SSP block media drivers, such as the SDMMC, SPI Flash and SDRAM/RAM. The interface remains the same for any media driver, so all media drivers appear functionally identical at the file I/O layer and can be interchanged with one another without changing code. Device adaptation drivers, such as the sf_block_media_ram, are accessed through the Block Media Framework Interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through block media function calls are generally device specific as well.

Block Media RAM Framework Module Important Operational Notes and Limitations

Block Media RAM Framework Module Operational Notes

The media must be formatted before you can open the media and begin creating, writing to and reading from files. The memory area used must be directly readable and writable by the core. Internal SRAM or SDRAM size should be sufficient for the Block Media RAM and there will be no

persistence of data across the power cycles. Block count must be assigned by considering the boot record, FAT area, root directory and directory sector. If the File System is used on RAM then the minimum block count must be 4, as the first 3 sectors are reserved for boot record, FAT area and root directory and the 4th sector is used for data sector. The "Format media during initialization" property needs to be enabled to use RAM for FileX. The "File System is on block media" property needs to be true inorder to configure the block count and block size from the block media RAM.

Block Media RAM Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.9.4 Including the Block Media RAM Framework Module in an Application

This section describes how to include the Block Media RAM Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media RAM Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Block Media RAM Framework is g_sf_block_media_ram0. This name can be changed in the associated Properties window.)

Block Media RAM Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_block_media_ram0 Block Media Framework on sf_block_media_ram	Threads	**New Stack> Framework> File System> Block Media Framework on sf_block_media_ram**

When the Block Media RAM Framework on sf_block_media_ram is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

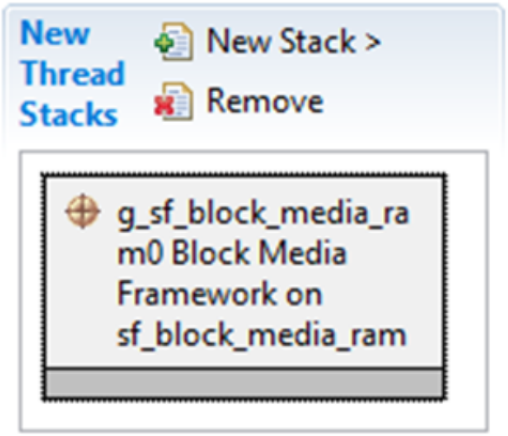


Figure 134: Block Media RAM Framework Module Stack

4.1.9.5 Configuring the Block Media RAM Framework Module

The Block Media RAM Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Block Media RAM Framework Module on sf_block_media_ram

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_block_media_ram0	Module name.
Block size of media in bytes	512	Block size of media in bytes selection.
Number of blocks to allocate	16	To make use of file system, block count must be assigned by considering the boot record, FAT area and root directory.
Enter the valid section for RAM buffer allocation	noinit	Enter the valid section for RAM buffer allocation.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Block Media RAM Framework Module Clock Configuration

The Block Media RAM Framework module uses the ICLK as the source for the internal RAM.

Block Media RAM Framework Module Pin Configuration

The Block Media RAM Framework module uses internal RAM so no external pins are required.

4.1.9.6 Using the Block Media RAM Framework Module in an Application

The steps in using the Block Media RAM Framework module on sf_block_media_ram in a typical application are:

1. Initialize the media using the FileX API, fx_system_initialize (sf_el_fx calls it automatically).
2. Format the media using FileX API, fx_media_format (sf_el_fx calls it automatically if "Format media during initialization" property is set to Enabled). This Format media is optional if media has already been formatted. Anything on the media will erase while media is formatted. (Optional)
3. Open the media using the FileX API fx_media_open (sf_el_fx opens the media automatically).
4. Read the media as required using the sf_block_media_api_t::read API (Block Media Framework) or one of the FileX APIs, for example, fx_media_read(), fx_file_read().
5. Write to the media as required using the sf_block_media_api_t::write API (Block Media Framework) or one of the FileX APIs, for example, fx_media_write(), fx_file_write().

These common steps are illustrated in a typical operational flow diagram in the following figure:

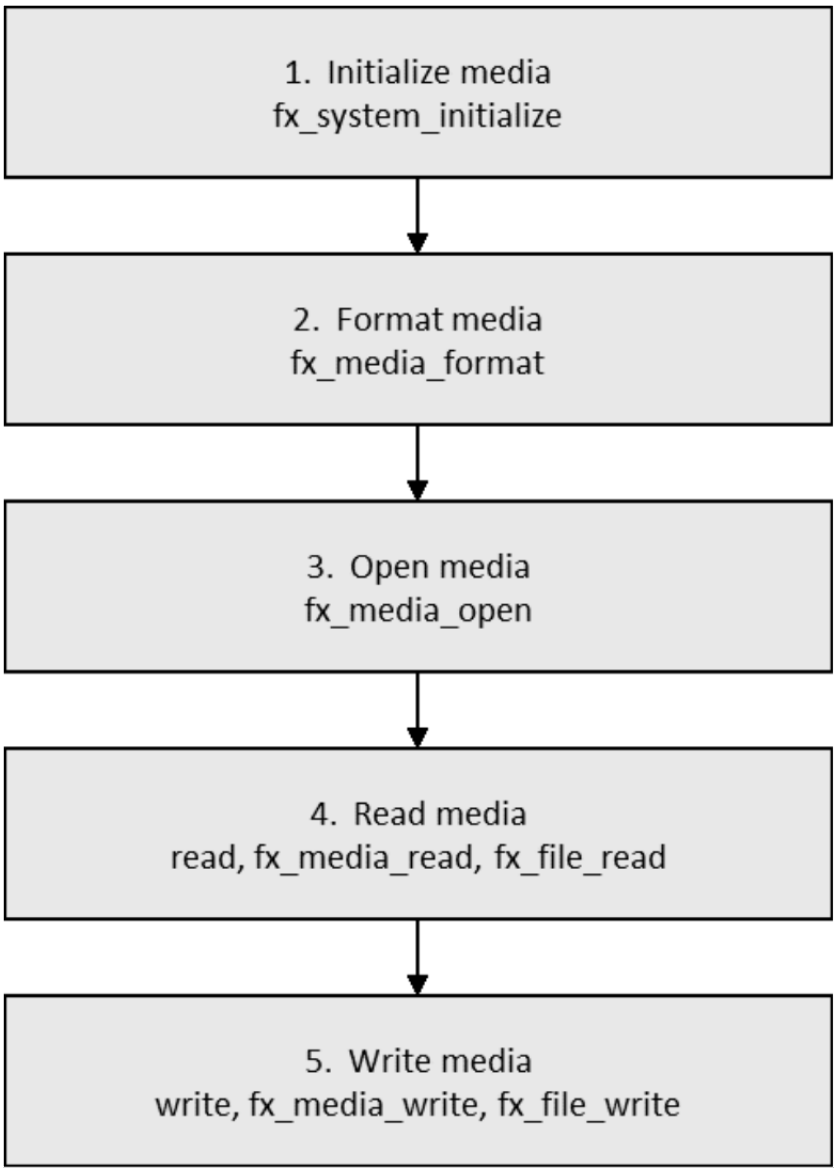


Figure 135: Flow Diagram of a Typical Block Media RAM Framework Module Application

4.1.10 Block Media Framework on sf_block_media_sdmmc

4.1.10.1 Block Media SDMMC Framework Module Introduction

The Block Media Framework module can implement the SD/MMC bus protocol for reading from, writing to and the control of SD cards and eMMC embedded devices through the SDHI (SD Host Interface) peripheral and the SD/MMC media driver. The driver has all the functionality needed to interface with a file system through a block media interface.

Block Media SDMMC Framework Module Features

- Supports SDHI host interface for SD/MMC.

- Supports SDSC (SD Standard Capacity), SDHC (SD High Capacity) and eMMC (embedded).
- Supports 1, 4 or 8-bit (eMMC only) data bus.

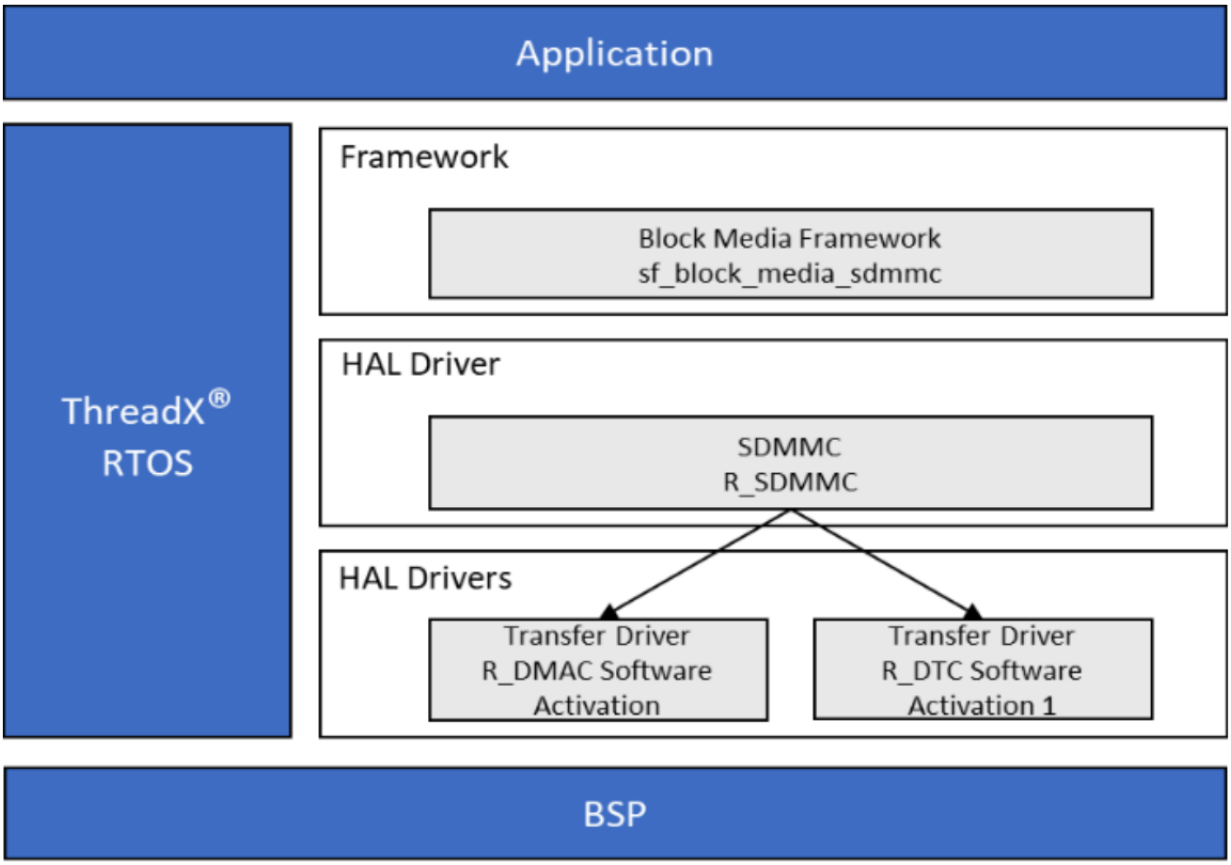


Figure 136: Block Media SDMMC Framework Module Block Diagram

4.1.10.2 Block Media SDMMC Framework Module APIs Overview

The Block Media Framework defines APIs for opening, reading from, writing to, controlling and closing the SDMMC. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media SDMMC Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_block_media_sdmmc_api->open(g_sf_block_media_sdmmc.p_ctrl, g_sf_block_media_sdmmc.p_cfg); Open device for read/write and control.
read	g_sf_block_media_sdmmc_api->read(g_sf_block_media_sdmmc.p_ctrl, &destination, startsector, sectorcount); Read data from SD/MMC.

write	g_sf_block_media_sdmmc_api->write(g_sf_block_media_sdmmc.p_ctrl, &source, startsector, sectorcount); Write data to SDMMC channel.
ioctl	g_sf_block_media_sdmmc_api->ioctl(g_sf_block_media_sdmmc.p_ctrl, command, &data); Send control commands to the SD/MMC port and receive the status of the SD/MMC port.
close	g_sf_block_media_sdmmc_api->close(g_sf_block_media_sdmmc.p_ctrl); Close open device port.
versionGet	g_sf_block_media_sdmmc_api->versionGet(&version); Get version of Block Media SD/MMC driver.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated control commands to reconfigure the channel.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_WRITE_PROTECTED	SD or MMC card is Write Protected.
SF_INFO_NOT_AVAILABLE	Information is not available possibly because card has been removed or is defective.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.10.3 Block Media SDMMC Framework Module Operational Overview

The Block Media Framework Interface is simply an abstract interface using function pointers instead of direct function calls. Functions are called between FileX and the SSP block media drivers, such as the SDMMC and the SPI Flash. The interface remains the same for any media driver, so all media drivers appear functionally identical at the file I/O layer and can be interchanged with one another without changing code. Device adaptation drivers, such as sf_block_media_sdmmc, are accessed through the Block Media Framework Interface and provide device specific code needed to perform

media I/O operations. Configuration and control structures passed through block media function calls are generally device specific as well.

Block Media SDMMC Framework Module Important Operational Notes and Limitations

Block Media SDMMC Framework Module Operational Notes

- The media must be formatted at least once before you can begin creating, writing and reading files.

Block Media SDMMC Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.10.4 Including the Block Media SDMMC Framework Module in an Application

This section describes how to include the Block Media SDMMC Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media SDMMC Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Block Media SDMMC Framework is g_sf_block_media_sdmmc0. This name can be changed in the associated Properties window.)

Block Media SDMMC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_block_media_sdmmc0 Block Media Framework on sf_block_media_sdmmc	Threads	**New Stack> Framework> File system> Block Media Framework**

When the Block Media SDMMC Framework on sf_block_media_sdmmc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

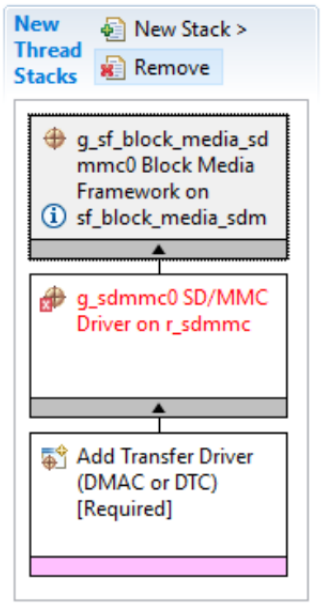


Figure 137: Block Media SDMMC Framework Module Stack

4.1.10.5 Configuring the Block Media SDMMC Framework Module

The Block Media SDMMC Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Block Media SDMMC Framework Module on sf_block_media_sdmmc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_block_media_sdmmc0	The name to be used for sf_block_media_sdmmc module control block instance.
Block size of media in bytes	512	Media Block size.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Block Media SDMMC Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the SDMMC HAL Module on r_sdmmc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_sdmmc0	The name to be used for SDMMC module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0, 1 Default: 1	Channel of SD/MMC peripheral, channel 0 or 1.
Media Type	Embedded, Card Default: Embedded	Media is a card or an embedded device. This allows to firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 Bit, 4 Bits, 8 Bits Default: 4 Bits	Data bus with as defined by hardware interface. (8 bits for eMMC only).
Block Size	512	Block size selection.
Card Detection	Not Used, CD Pin Default: CD Pin	Card detection selection.
Callback	NULL	(Not required if using Filex) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR

Access Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Access interrupt priority selection.
Card Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Card interrupt priority selection.
DMA Request Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	DMA request interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build.
Name	g_transfer0	Module name.
Channel	0	
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation	Activation source selection.
Auto Enable	False	Auto enable selection.

Callback (Only valid with Software start)	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Block Media SDMMC Framework Module Clock Configuration

The SDHI block (used to implement SDMMC and SDIO functions) uses the PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO or eMMC device, obtained at media device initialization.

Block Media SDMMC Framework Module Pin Configuration

Use the e^2 studio pin configurator to configure the I/O pins for the SDMMC peripheral. The drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the Block Media SDMMC Framework Module on sf_block_media_sdmmc

Resource	ISDE Tab	Pin selection Sequence
SDHI	Pins	Select **Peripherals**> **Storage:SHDI**> **SDHI0**

Note

The selection sequence assumes SCI1 is the desired hardware target for the driver.

Pin Configuration Settings for the Block Media SDMMC Framework Module on sf_block_media _sdmmc

Property	Value	Description
Operation Mode	Disabled, Custom, SD_MMC 1 bit SD_MMC 4 bit MMC 8 bit	Select mode as per application.
CLK	None, P413 Default: None	Clock Pin.

CMD	None, P412 Default: None	Command Pin.
DAT0-7	None, PXXX Default: None	Data Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

Other Settings

The read and write media and extended read and write SDIO functions are non-blocking and require interrupts and a transfer function, either DMAC or DTC. The read and write functions return SSP_SUCCESS to indicate that the initial operations have started successfully. However, the user application must wait for the user callback and check for event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

4.1.10.6 Using the Block Media SDMMC Framework Module in an Application

The steps in using the Block Media SDMMC Framework module on sf_block_media_sdmmc in a typical application are:

1. Initialize media using the sf_block_media_api_t::open API.
2. Read media as required using the sf_block_media_api_t::read API.
3. Write media as required using the sf_block_media_api_t::write API.
4. Operate on data as required.
5. Close media as required using the sf_block_media_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

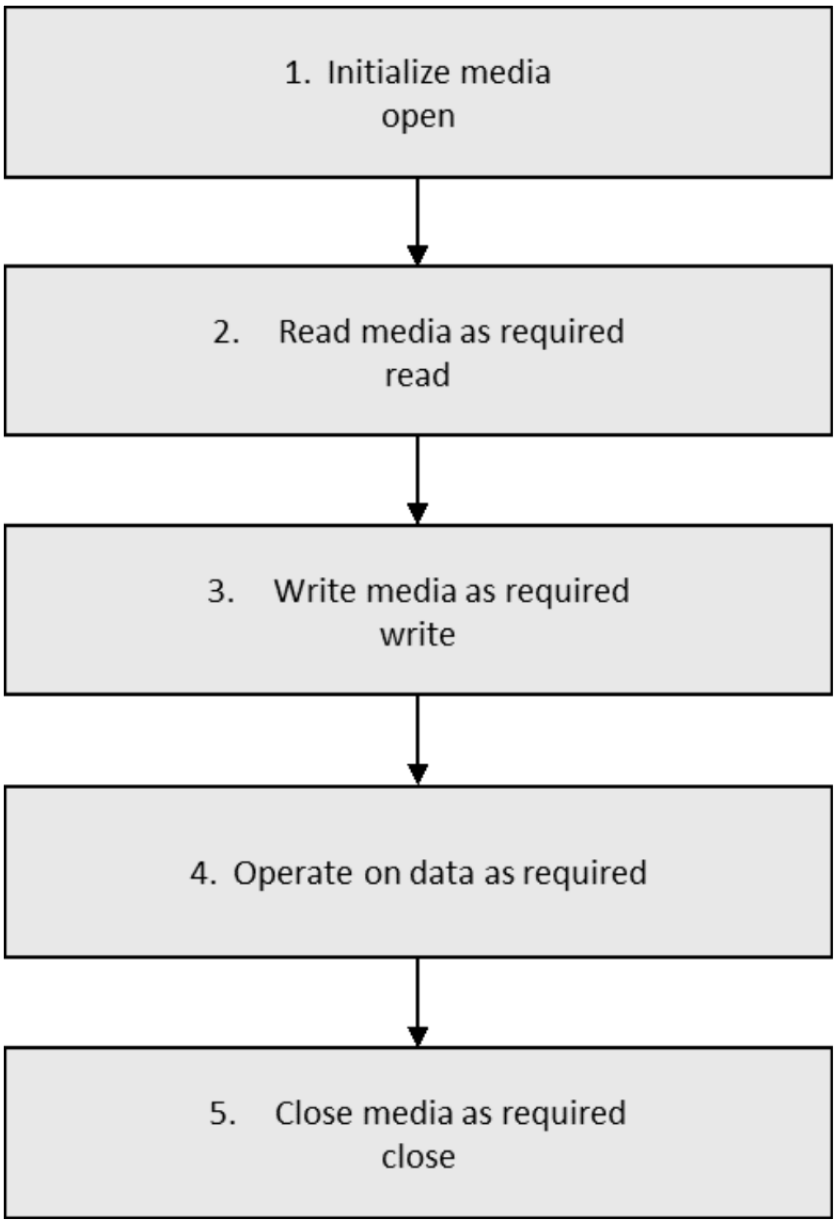


Figure 138: Flow Diagram of a Typical Block Media SDMMC Framework Module Application

The typical steps for using the sf_block_media_sdmmc using the sf_el_fx in an application are:

1. Initialize media using the FileX API, fx_system_initialize (sf_el_fx calls it automatically)
2. Format media using FileX API, fx_media_format (sf_el_fx calls it automatically if "Format media during initialization" property is set to Enabled)
3. Open the media using FileX API, fx_media_open (sf_el_fx opens the media automatically)
4. Read media as required using the sf_block_media_api_t::read API (Block Media Framework) or one of the FileX API, for example, fx_media_read(), fx_file_read()
5. Write to media as required using the sf_block_media_api_t::write API (Block Media Framework) or one of the FileX API, for example, fx_media_write(), fx_file_write().

Note

After successful fx_media_open call, all FileX APIs can be used (not only read and write).

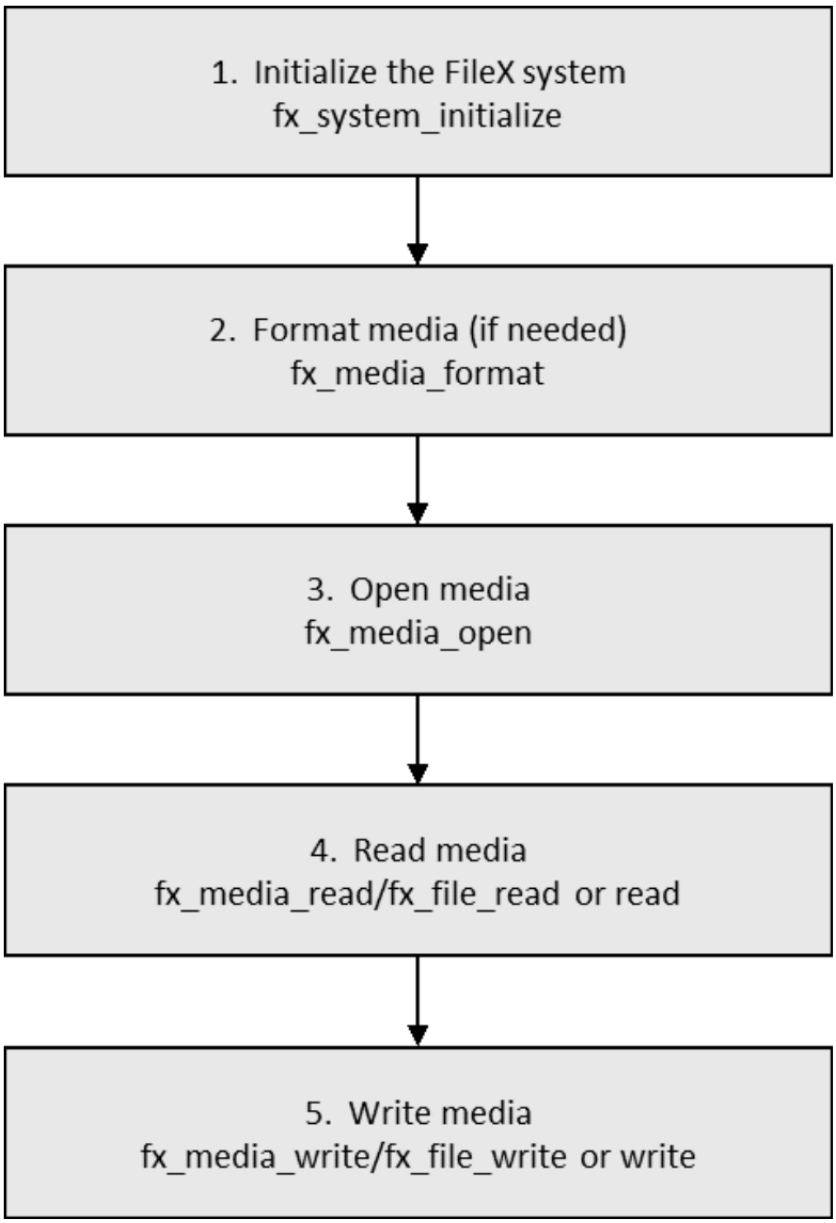


Figure 139: Flow Diagram of a Typical Block Media SDMMC Framework Module Application

4.1.11 BLE Framework

4.1.11.1 BLE Framework Introduction

Bluetooth® Low Energy (BLE), sometimes referred to as Bluetooth Smart, is a light-weight subset of Classic Bluetooth, and was introduced as part of the Bluetooth 4.0 core specification. In contrast to Classic Bluetooth, BLE is designed to provide significantly lower power consumption. This allows Internet of Thing (IoT) devices that have stricter power capacity to transfer small amounts of data between nearby devices.

Application developers access the functionality provided by the BLE stack using its APIs. The BLE stack APIs provided by different vendors are not standardized, and as a result, Application developers have to update their code when porting to different BLE stacks.

The Synergy BLE Framework handles this issue by providing a generic interface for the underlying BLE stack provided by various vendors, thereby preventing coupling between application and vendor-specific BLE stack code. The use of generic APIs makes application development simpler and portable.

The BLE Framework provides a high-level API for BLE applications and uses the Synergy Software Package (SSP) communication framework, which in turn enables the UART driver for communication to the underlying BLE module. It also integrates the generic BLE profile framework (g_sf_ble_onboard_profile), which provides a uniform interface to BLE profiles. For the RL78G1D BLE hardware module, the generic BLE profiles are implemented by the BLE module firmware.

BLE Framework Module Features

- ThreadX® RTOS Aware and thread safe
- Bluetooth v4.2 compliant framework
- Generic Access Profile (GAP) Features
    - User-defined advertising data
    - Security modes 1 and 2
    - Peripheral and central roles
    - White list support for up to 6 devices
    - Bonding support
- Generic Attribute Profile (GATT) features
    - GATT client and server
- Generic Attribute Profile (GATT) APIs
- Generic Access Profile (GAP) APIs
- Generic On-board Profiles APIs

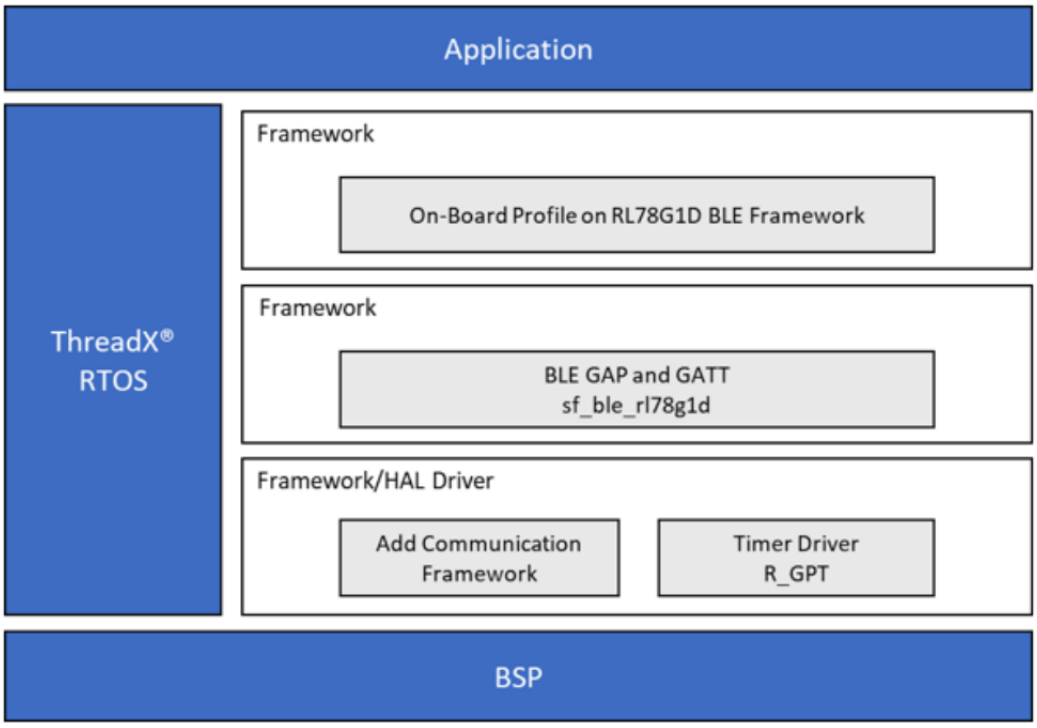


Figure 140: BLE Framework Module Block Diagram

Note

The BLE and GATT on sf_ble_rl78g1d can be used as a lower-level implementation of the On-Board Profile on RL789G1D BLE Framework or on its own.

4.1.11.2 BLE Framework Module APIs Overview

The BLE Framework defines APIs for initializing, setting and getting values, and stopping modules. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

BLE Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_ble0.p_api->open(g_ sf_ble_0.p_cfg); This function initializes and enables the BLE module. It accepts the BLE module configuration as an argument, which has the following parameters: - 48-bit Bluetooth Address - Device scan interval - Device scan window - Device discoverable time - Device connection interval - Slave latency - Supervision timeout - Own address type - Maximum slaves allowed to be connected
close	g_sf_ble0.p_api->close (g_ sf_ble0.p_cfg); This API de-initializes the interface and may put the BLE module in low power mode or power it off. It also closes the driver, disables the driver link, disables the interrupt in the BLE module driver.
infoGet	g_sf_ble0.p_api->infoGet (g_sf_ble_0.p_cfg, p_handle, p_ble_info); The infoGet API takes the BLE control structure as an argument. It returns the following information obtained from the BLE module: - Chipset/driver information string - RSSI value (unsigned integer 16 bits)
provisionGet	g_sf_ble0.p_api->provisionGet (g_sf_ble_0.p_cfg, p_ble_provisioning); The provisionGet API gets the BLE GAP provisioning information and takes the BLE control structure as an argument. It returns the following parameters: - Bonding Mode - Security Mode - GAP Role (Central/Master or Peripheral/Slave) - Security Keys

provisionSet	g_sf_ble0.p_api->provisionSet (g_sf_ble_0.p_cfg, p_ble_provisioning); The provisionSet() function provisions the BLE module. It takes the BLE control structure and the provisioning structure as an argument. - Bonding Mode - Security Mode - GAP Role (Central/Master or Peripheral/Slave) - Security Keys - GAP user event callback
scan	g_sf_ble0.p_api->scan (g_sf_ble_0.p_cfg, p_scan, p_cnt, p_scan_info); The scan() function takes BLE control structure as an argument. The scan() function returns a list of BLE devices scanned by the BLE module with below parameters. - Bluetooth address (48-bits) - RSSI - Scan data - Advertising Event type The scan() function takes device count as an argument, which acts as an in/out parameter. It specifies the size of the scan result array and the BLE framework sets it to count indicating the number of scan results stored in the array. The function takes scan type as an argument (active/passive).
advertisementStart	g_sf_ble0.p_api->advertisementStart (g_sf_ble_0.p_cfg, p_advt_info); The advertisementStart() function takes the following parameters: - Discovery mode (General/Limited) - Filter policy – Support for scan/connect request filtering combinations - Advertisement data - Connection mode - Advertisement intervals - Channel map - Address type - Advertising type - Scan response data
advertisementStop	g_sf_ble0.p_api->advertisementStop (g_sf_ble_0.p_cfg); Stops advertisement.
whitelistAdd	g_sf_ble0.p_api->whitelistAdd (g_sf_ble_0.p_cfg, p_bp_addr); The whitelistAdd() function adds devices to the whitelist for advertisements, scan requests and connection requests.

whitelistDel	g_sf_ble0.p_api->whitelistDel (g_sf_ble_0.p_cfg, p_bp_addr); The whitelistDel() function deletes devices from the whitelist for advertisements, scan requests and connection requests.
bondingStart	g_sf_ble0.p_api->bondingStart (g_sf_ble_0.p_cfg, p_handle, p_bp_addr, p_bonding_start); The bondingStart() function starts bonding with a remote device.
bondingResponse	g_sf_ble0.p_api->bondingResponse (g_sf_ble_0.p_cfg, p_handle, p_bp_addr, p_bonding_resp); The bondingResponse() function responds to a bonding request.
startEncryption	g_sf_ble0.p_api->startEncryption (g_sf_ble_0.p_cfg, p_enc_info); The startEncryption() function begins an encryption operation.
connect	g_sf_ble0.p_api->connect (g_sf_ble_0.p_cfg, p_handle, p_conn); The connect() function connects to a remote device.
disconnect	g_sf_ble0.p_api->disconnect (g_sf_ble_0.p_cfg, p_handle); The disconnect() function disconnects from a remote device.
listen	g_sf_ble0.p_api->listen (g_sf_ble_0.p_cfg); The listen() function listens for an incoming connect request from aremote device.
authorization	g_sf_ble0.p_api->authorization (g_sf_ble_0.p_cfg, &conhandle); The authorization() function authorizes a remote device after connection.
setTxPower	g_sf_ble0.p_api->setTxPower(g_sf_ble_0.p_cfg, &con_handle, &tx_power_info); The setTxPower() function sets the transmit power for the procedure specified by the connection handle.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

BLE Framework On-Board Profiles Module API Summary

Function Name	Example API Call and Description

open	g_sf_ble_onboard_profile0.p_api->open (g_sf_ble_onboard_profile0.p_cfg); This API initializes the interface for data transfers.
close	g_sf_ble_onboard_profile0.p_api->close (g_sf_ble_onboard_profile0.p_cfg); This API de-initializes the interface and may put it in low power mode or power it off. The API closes the driver, and disables the driver link and interrupt.
onbpEnable	g_sf_ble_onboard_profile0.p_api->onbpEnable (sf_ble_onboard_profile0.p_cfg, p_handle, profile, p_prf_cb, sec); Enables the profile in server mode or client mode.
onbpServerWriteData	g_sf_ble_onboard_profile0.p_api->onbpServerWriteData (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics, p_data); Updates the value of the characteristic in the local database.
onbpServerSendNotification	g_sf_ble_onboard_profile0.p_api->onbpServerSendNotification (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics, p_data); Sends notifications.
onbpServerSendIndication	g_sf_ble_onboard_profile0.p_api->onbpServerSendIndication (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics, p_data); Sends indications.
onbpClientWriteCCCD	g_sf_ble_onboard_profile0.p_api->onbpClientWriteCCCD (sf_ble_onboard_profile0.p_cfg, p_handle, profile, cccd_char, cccd_val); Sets the Client Configuration Control Descriptor on the remote device.
onbpDisable	g_sf_ble_onboard_profile0.p_api->onbpDisable (sf_ble_onboard_profile0.p_cfg, p_handle, profile); Disables the profile in server mode and clientmode.
onbpClientReadChar	g_sf_ble_onboard_profile0.p_api->onbpClientReadChar (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics); Reads a GATT characteristic associated with the profile or service.
onbpClientWriteChar	g_sf_ble_onboard_profile0.p_api->onbpClientWriteChar (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics); Writes a GATT characteristic associated with the profile or service.

versionGet	g_sf_ble_onboard_profile0.p_api-> versionGet(&version); Retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.
All the details related to BLE standard profiles can be found in BLE profile specifications.
BLE APIs will return SSP_ERR_UNSUPPORTED if the module does not support the feature.

BLE Framework GATT API Summary

Function Name	Example API Call and Description
gattAddCustomProfiles	g_sf_ble0.p_api->gattAddCustomProfiles (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); This function adds custom profiles to the GATT database.
gattServiceDiscovery	g_sf_ble0.p_api->gattServiceDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattServiceDiscovery() function performs service discovery.
gattCharDiscovery	g_sf_ble0.p_api->gattCharDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattCharDiscovery() function performs the Char discovery operation.
gattCharDescDiscovery	g_sf_ble0.p_api->gattCharDescDiscovery (g_sf_ble_0.p_cfg, p_handle, start_handle, end_handle, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); Discovers GATT characteristics descriptor on a remote device.
gattCharRead	g_sf_ble0.p_api->gattCharRead (g_sf_ble_0.p_cfg, p_handle, start_handle, p_char_read_req, p_char_read_rsp); Reads GATT characteristics on a remote device.
gattCharWrite	g_sf_ble0.p_api->gattCharWrite (g_sf_ble_0.p_cfg, p_handle, p_char_read_req); Writes GATT characteristics on a remote device.
gattCharExecuteWrite	g_sf_ble0.p_api->gattCharExecuteWrite (g_sf_ble_0.p_cfg, p_handle, execute_flag); Executes a write (commit) on GATT characteristics on a remote device.

gattCharWriteLocal	g_sf_ble0.p_api->gattCharWriteLocal (g_sf_ble_0.p_cfg, char_handle, data_length); Updates the local GATT database.
gattSendNotify	g_sf_ble0.p_api->gattSendNotify (g_sf_ble_0.p_cfg, p_handle, char_handle); Sends notifications from local GATT server to remote GATT client.
gattSendIndicate	g_sf_ble0.p_api->gattSendIndicate (g_sf_ble_0.p_cfg, p_handle, char_handle); Sends indications from local GATT server to remote GATT client.
gattWriteResponse	g_sf_ble0.p_api->gattWriteResponse (g_sf_ble_0.p_cfg, p_handle, char_handle); Responds to the write characteristic value request from the remote GATT client.
versionGet	g_sf_ble0.p_api->versionGet(&version); Retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_PTR	p_version is NULL.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.11.3 BLE Framework Module Operational Overview

This section provides the Synergy BLE Framework software architecture overview and highlights the major SSP modules used as part of BLE framework along with the operational flow sequence from the user's application level.

Note

A more comprehensive description of the operation of the BLE Framework module is available in the BLE Framework Application Project. The complete project and associated application note can be found by doing a search for "r30qan0309eu" in the search bar at the top of the www.renesas.com home page.

BLE Framework Module Important Operational Notes and Limitations

BLE Framework Module Operational Notes

The BLE framework provides a common interface for the application. The implementation of the interface is specific for each module. The Synergy BLE framework currently defines an interface implemented for RL78G1D BLE module. Each implementation interacts with the corresponding BLE device driver. The BLE device driver uses the underlying SSP communication framework (g_sf_comms), which in turn interacts with the SSP HAL components such as Universal Asynchronous Receiver/Transmitter (UART), Data Transfer Controller (DTC), and General PWM Timer (GPT) drivers to communicate with the BLE module. The following figure shows a high-level architectural description of the BLE Framework module.

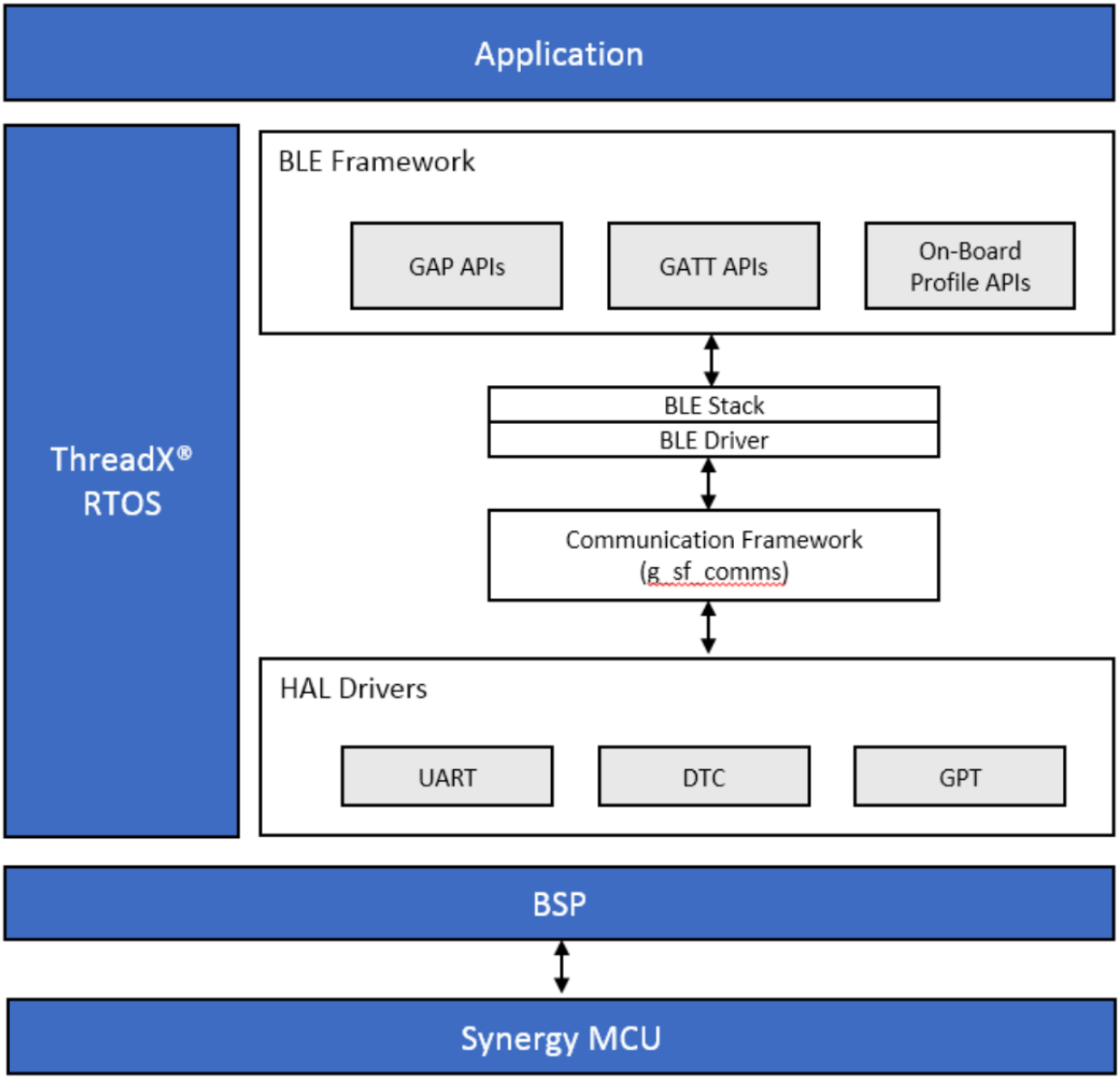


Figure 141: Typical BLE Module Architecture Types

GAP and GATT APIs

The BLE framework provides a generic interface for the application to configure and provision the BLE module. The BLE module has various configuration parameters as specified by the family of Bluetooth Smart standards. It is possible that individual device drivers and/or BLE modules might not support all configuration parameters. At a bare minimum, the provisioning API provides a mechanism

to set the operating mode, security mode, security keys, and bonding mode of the BLE interface. It also provides an API for the GAP/GATT layers.

On-Board Profiles APIs

The on-board profiles APIs provide a uniform interface to the BLE profiles implemented by the BLE module firmware.

BLE Stack

The BLE module host stack is typically provided by the BLE module vendor. The BLE module typically comes in three different flavors depending on the HW/SW partitioning between the host MCU and BLE module. The RL78G1D BLE module is part of the Network Controller Implementation architecture, where the BLE chipset includes all the implementation for the BLE link layer, GAP, GATT, and on-board profiles. The module interfaces with the MCU over the sf_comms framework provided by SSP.

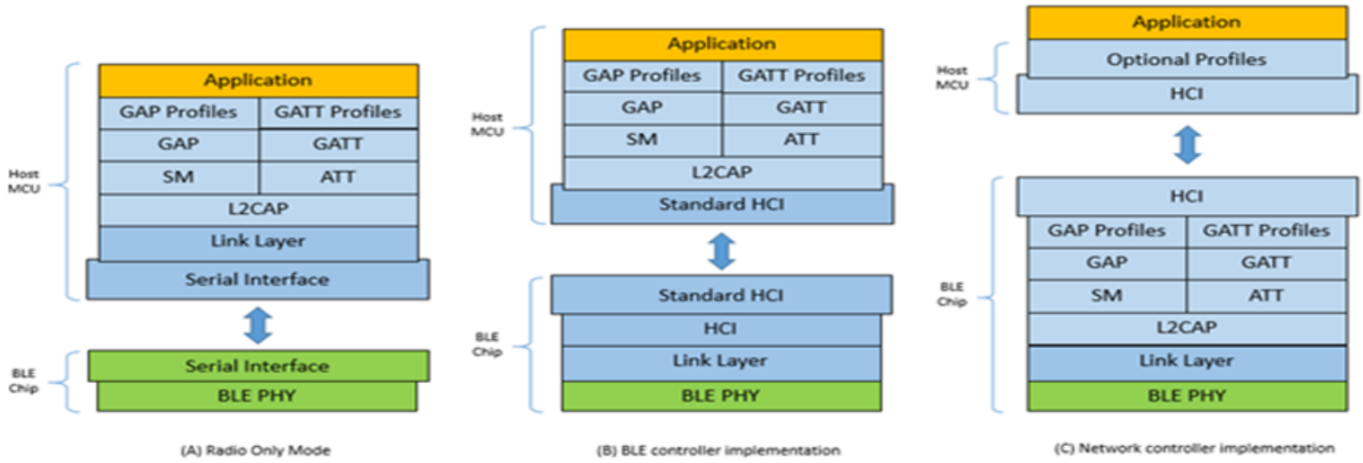


Figure 142: BLE Module Architecture Types

A: BLE radio-only mode

Link layer, L2CAP, GATT, GAP layers, profiles, and application run on the host MCU. Physical layer runs on BLE chipset.

B: BLE controller implementation

Link layer runs on BLE chipset, L2CAP, and higher BLE protocol (GATT, GAP) layers. Profiles and application run on the host MCU.

C: Network controller implementation

Link layer, L2CAP, GATT, GAP layers, and generic profiles run on the BLE chipset. Optional profiles and application run on the host processor.

BLE Framework Instances

An application must define the BLE Framework instance before using it. The instance is a structure that includes pointers to any of the following:

BLE Framework Control Structure

This structure is used in all BLE Framework APIs. This structure includes pointer to driver handle, which is used by the framework for storing the required information by the BLE device driver.

BLE Framework Configuration Structure

This structure is passed to sf_ble_api_t::open API and you can use this structure to configure the BLE module. This configuration is applied either during initialization, such as open or provisioning such as sf_ble_api_t::provisionSet API. Configuration parameters that are not supported by the BLE module are ignored by the framework.

BLE Framework APIs Structure

This structure contains pointers to the BLE Framework APIs that are specific to a given module. See Configuring the BLE Framework Module for more details on these APIs.

BLE Framework Module Operational Flow

The steps for using the BLE Framework module in an application are:

1. Initialize the BLE hardware module.
2. Select the GATT layer role such as GATT client or GATT server. It is most common for the slave (peripheral) device to be the GATT server and the master (central) device to be the GATT client.
3. The application controls operations using generic (on-board) profile APIs or GAP/GATT APIs.

Note

The GAP provisioning structure has a BLE user callback that runs in the driver thread context. An application should make sure that callback logic is as minimal as possible without any blocking calls. Print statements or blocking calls may introduce delays in BLE driver execution. Make sure that no BLE APIs are called in user callbacks as it may also lead to code failure.

BLE Module Initialization Flow Sequence

The following BLE module initialization sequence is part of the Synergy auto-generated code:

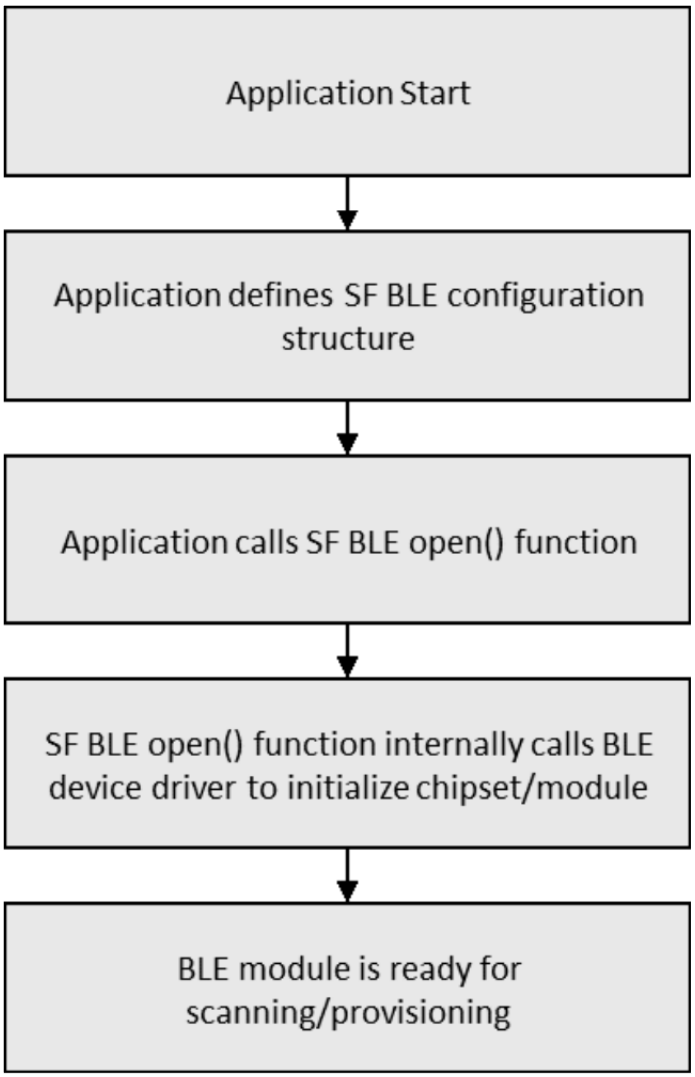


Figure 143: Auto-Generated Initialization Sequence Flow Chart

On-Board Profile-Based Client Application Flow Sequence

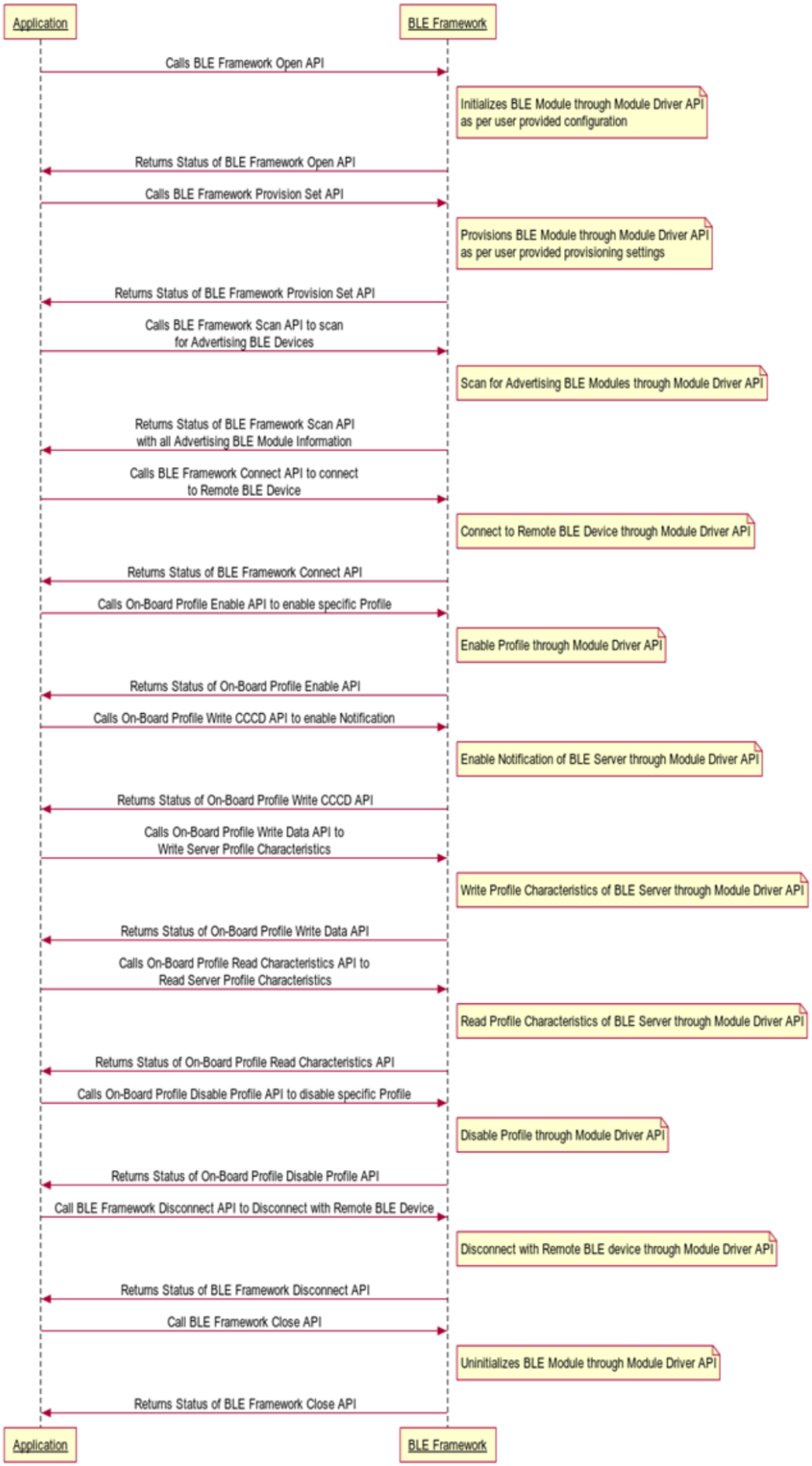

Application
BLE Framework
Calls BLE Framework Open API
Initializes BLE Module through Module Driver API as per user provided configuration
Returns Status of BLE Framework Open API
Calls BLE Framework Provision Set API
Provisions BLE Module through Module Driver API as per user provided provisioning settings
Returns Status of BLE Framework Provision Set API
Calls BLE Framework Scan API to scan for Advertising BLE Devices
Scan for Advertising BLE Modules through Module Driver API
Returns Status of BLE Framework Scan API with all Advertising BLE Module Information
Calls BLE Framework Connect API to connect to Remote BLE Device
Connect to Remote BLE Device through Module Driver API
Returns Status of BLE Framework Connect API
Calls On-Board Profile Enable API to enable specific Profile
Enable Profile through Module Driver API
Returns Status of On-Board Profile Enable API
Calls On-Board Profile Write CCCD API to enable Notification
Enable Notification of BLE Server through Module Driver API
Returns Status of On-Board Profile Write CCCD API
Calls On-Board Profile Write Data API to Write Server Profile Characteristics
Write Profile Characteristics of BLE Server through Module Driver API
Returns Status of On-Board Profile Write Data API
Calls On-Board Profile Read Characteristics API to Read Server Profile Characteristics
Read Profile Characteristics of BLE Server through Module Driver API
Returns Status of On-Board Profile Read Characteristics API
Calls On-Board Profile Disable Profile API to disable specific Profile
Disable Profile through Module Driver API
Returns Status of On-Board Profile Disable Profile API
Call BLE Framework Disconnect API to Disconnect with Remote BLE Device
Disconnect with Remote BLE device through Module Driver API
Returns Status of BLE Framework Disconnect API
Call BLE Framework Close API
Uninitializes BLE Module through Module Driver API
Returns Status of BLE Framework Close API
Application
BLE Framework

Figure 144: On-Board Profile Client Application Flow

On-Board Profile-Based Server Application Flow Sequence

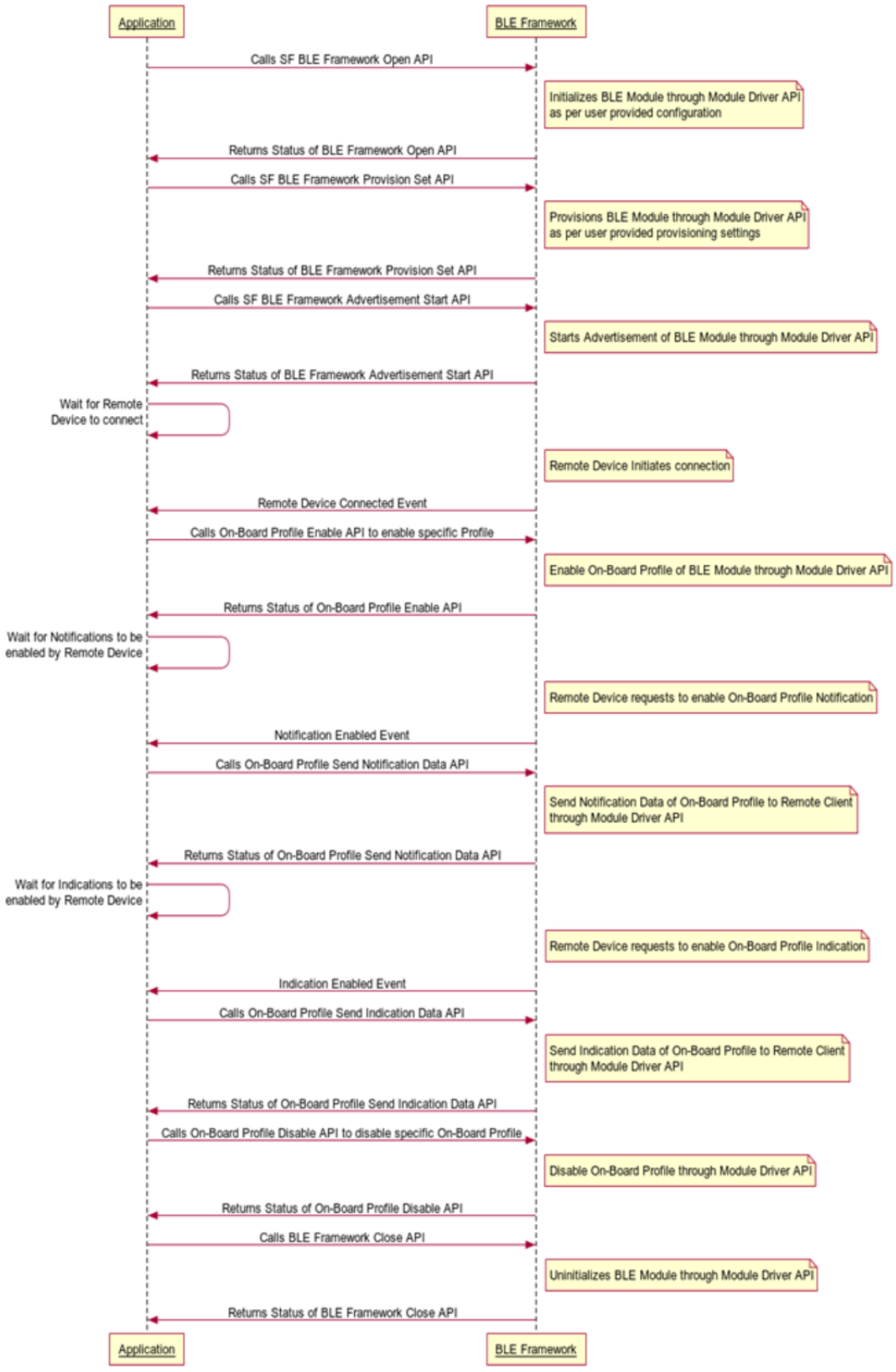


Figure 145: On-Board Profile Server Application Flow

GAP/GATT-Based Client Application Flow Sequence

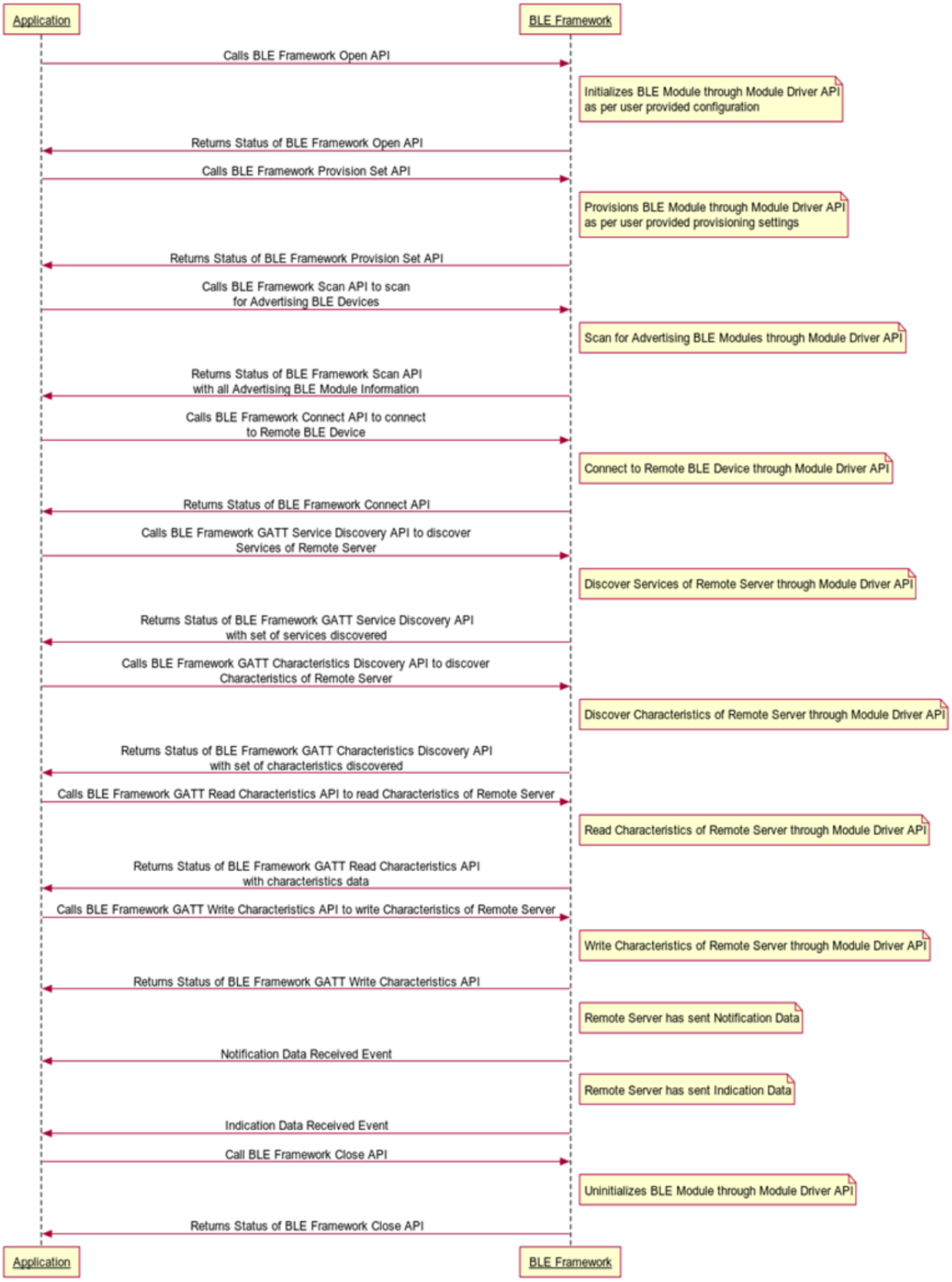

Application
BLE Framework
Calls BLE Framework Open API
Initializes BLE Module through Module Driver API as per user provided configuration
Returns Status of BLE Framework Open API
Calls BLE Framework Provision Set API
Provisions BLE Module through Module Driver API as per user provided provisioning settings
Returns Status of BLE Framework Provision Set API
Calls BLE Framework Scan API to scan for Advertising BLE Devices
Scan for Advertising BLE Modules through Module Driver API
Returns Status of BLE Framework Scan API with all Advertising BLE Module Information
Calls BLE Framework Connect API to connect to Remote BLE Device
Connect to Remote BLE Device through Module Driver API
Returns Status of BLE Framework Connect API
Calls BLE Framework GATT Service Discovery API to discover Services of Remote Server
Discover Services of Remote Server through Module Driver API
Returns Status of BLE Framework GATT Service Discovery API with set of services discovered
Calls BLE Framework GATT Characteristics Discovery API to discover Characteristics of Remote Server
Discover Characteristics of Remote Server through Module Driver API
Returns Status of BLE Framework GATT Characteristics Discovery API with set of characteristics discovered
Calls BLE Framework GATT Read Characteristics API to read Characteristics of Remote Server
Read Characteristics of Remote Server through Module Driver API
Returns Status of BLE Framework GATT Read Characteristics API with characteristics data
Calls BLE Framework GATT Write Characteristics API to write Characteristics of Remote Server
Write Characteristics of Remote Server through Module Driver API
Returns Status of BLE Framework GATT Write Characteristics API
Remote Server has sent Notification Data
Notification Data Received Event
Remote Server has sent Indication Data
Indication Data Received Event
Call BLE Framework Close API
Uninitializes BLE Module through Module Driver API
Returns Status of BLE Framework Close API
Application
BLE Framework

Figure 146: GAP/GATT Client Application Flow

GAP/GATT-Based Server Application Flow Sequence

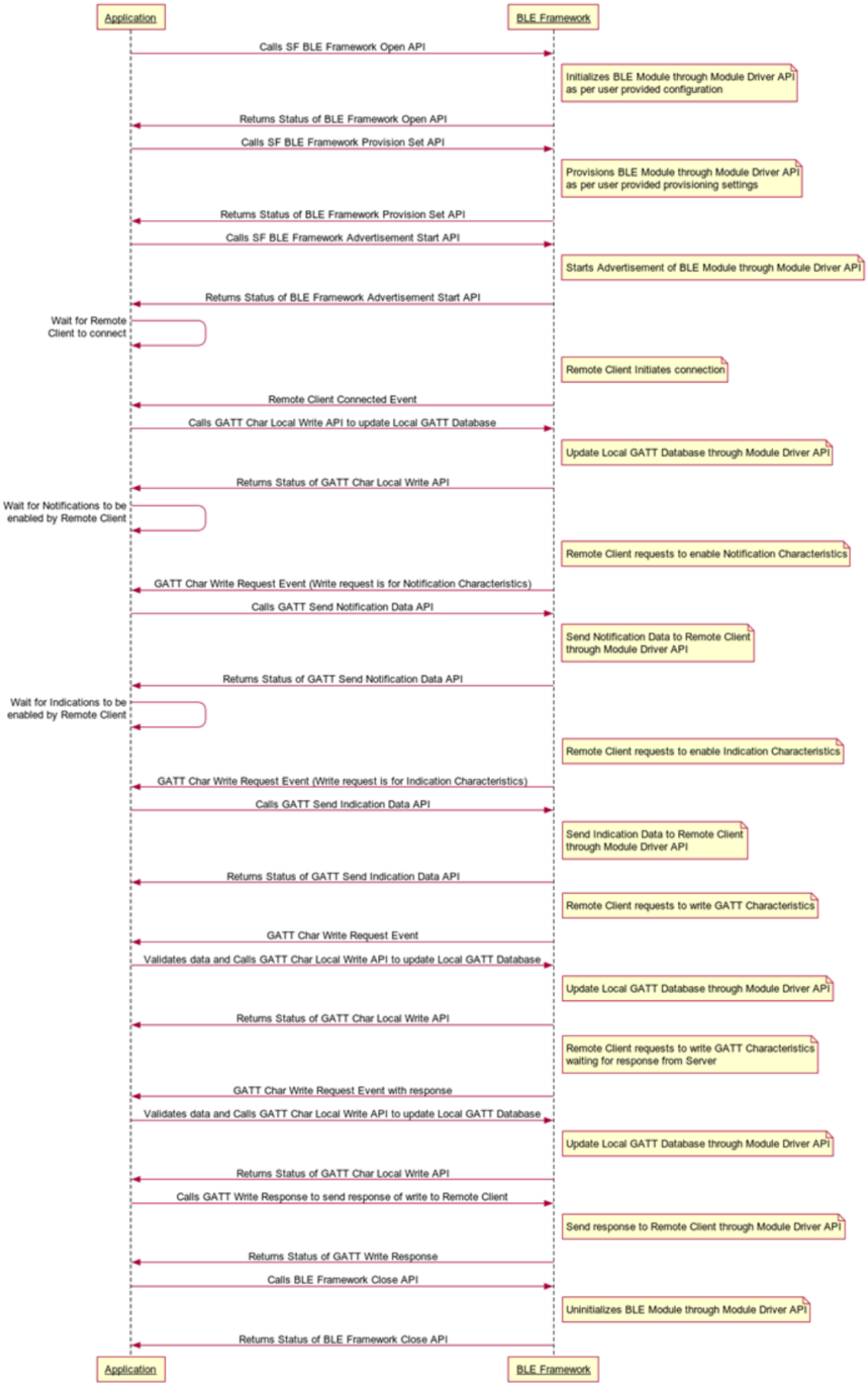

Application
BLE Framework
Calls SF BLE Framework Open API
Initializes BLE Module through Module Driver API as per user provided configuration
Returns Status of BLE Framework Open API
Calls SF BLE Framework Provision Set API
Provisions BLE Module through Module Driver API as per user provided provisioning settings
Returns Status of BLE Framework Provision Set API
Calls SF BLE Framework Advertisement Start API
Starts Advertisement of BLE Module through Module Driver API
Returns Status of BLE Framework Advertisement Start API
Wait for Remote Client to connect
Remote Client Initiates connection
Remote Client Connected Event
Calls GATT Char Local Write API to update Local GATT Database
Update Local GATT Database through Module Driver API
Returns Status of GATT Char Local Write API
Wait for Notifications to be enabled by Remote Client
Remote Client requests to enable Notification Characteristics
GATT Char Write Request Event (Write request is for Notification Characteristics)
Calls GATT Send Notification Data API
Send Notification Data to Remote Client through Module Driver API
Returns Status of GATT Send Notification Data API
Wait for Indications to be enabled by Remote Client
Remote Client requests to enable Indication Characteristics
GATT Char Write Request Event (Write request is for Indication Characteristics)
Calls GATT Send Indication Data API
Send Indication Data to Remote Client through Module Driver API
Returns Status of GATT Send Indication Data API
Remote Client requests to write GATT Characteristics
GATT Char Write Request Event
Validates data and Calls GATT Char Local Write API to update Local GATT Database
Update Local GATT Database through Module Driver API
Returns Status of GATT Char Local Write API
Remote Client requests to write GATT Characteristics waiting for response from Server
GATT Char Write Request Event with response
Validates data and Calls GATT Char Local Write API to update Local GATT Database
Update Local GATT Database through Module Driver API
Returns Status of GATT Char Local Write API
Calls GATT Write Response to send response of write to Remote Client
Send response to Remote Client through Module Driver API
Returns Status of GATT Write Response
Calls BLE Framework Close API
Uninitializes BLE Module through Module Driver API
Returns Status of BLE Framework Close API
Application
BLE Framework

Figure 147: GAP/GATT Client Application Flow

BLE Framework Security

Security Manager provides BLE protocol stack the ability to generate and exchange security keys which is used to encrypt communication link. The Security Manager has two functions:

- Initiator - This is the GAP Master/Central device
- Responder - This is the GAP Slave/Peripheral device.

The initiator is the master device that initiate the security procedure, however the slave device can asynchronously request the initiator to begin the security procedure.

BLE Framework Security Modes

BLE Security provides modes with levels associated with each mode. Security mode and level is a combination of support for authenticated or unauthenticated pairing, encryption or data signing. Pairing is required to satisfy various security requirements. Two types of pairing are available:

Authenticated pairing where devices are protected from MITM (Man In The Middle) attacks

Unauthenticated pairing where they are not protected from MITM.

Security Mode 1

- Security Level 1: No Security
- Security Level 2: Unauthenticated pairing with encryption
- Security Level 3: Authenticated pairing with encryption
- Security Level 4: Authenticated LE secure connections pairing with encryption

Security Mode 2

- Security Level 1: Unauthenticated pairing with data signing
- Security Level 2: Authenticated pairing with data signing

Note

RL78G1D BLE module does not support Security Mode 1 with Security Level 4.

On-Board Profile Security Modes

No Security: If On-Board Profile security is set to No Security then Profile communication works in unsecured mode regardless of BLE GAP Security Mode and Level.

Unauthenticated: For this Profile Security method to work any of the BLE GAP Security Method other than Security Mode 1 Level 1 should be used. Module should have completed Pairing with the remote device.

Authenticated: For this Profile Security method to work any of the following BLE GAP Security Method should be used. Module should have completed Pairing with the remote device.

- Security Mode 1 Level 3
- Security Mode 2 Level 2

Authorization: For this Profile Security method to work the remote device should have been Authorized during GAP connection using BLE Framework authorization API. This On-Board profile security parameter is specific to server API only.

Encryption: In this Security procedure, the profile will use encrypted communication. For this Profile Security method to work, any of the BLE GAP Security methods other than Security Mode 1 Level 1 should be used. If any security method that does not use encryption is used, then the profile works in unencrypted mode.

1. On-Board Profile security can be set in the sf_ble_onboard_profile_api_t::onbpEnable function. The sf_ble_onboard_profile_api_t::onbpEnable API enables the profile in server mode or client mode. It is a generic API to enable client and server mode. But, the profile security parameter is specific to server API only. Security parameter can be set for server only in driver APIs. So, user needs to set profile security for server enable API.
2. Profile security enums are in bit pattern, so more than one profile security can be set using bitwise operators. There are a few points to keep in mind while setting the profile security:
    - If profile security is none, then other security bits should not be set.
    - Security bits should not be set to both authenticated and unauthenticated.

BLE Framework Security Procedure

BLE Security has the following procedures:

- Pairing: This procedure is used to generate temporary encryption key to encrypt communication link. Permanent encryption keys can be shared over this encrypted communication link for additional communication.
- Bonding: This is a combination of pairing and storing of permanent keys. After pairing, the permanent keys are stored in a non-volatile memory, which creates a permanent bond between two devices. For subsequent communication, it is not necessary for devices to perform the bonding procedure.
- Encryption Establishment: Communication is encrypted using permanent keys.

Pairing creates a secure link that lasts for the lifetime of the connection, whereas bonding creates a permanent association called bond.

BLE Security Phases

BLE Security goes through three phases. Two devices establish connection using the GAP connection procedure, followed by the three phases to establish a secure communication link:

- Phase 1 (Pairing Phase, Information Sharing): Initially in phase 1, all information required to generate the temporary keys are shared between two devices.
- Phase 2 (Pairing Phase, Temporary Key Sharing): In this phase, temporary encryption key (Short Term Key or STK) is generated on both devices. This is used to encrypt the connection. This encrypted link can be used for additional communication. This communication link remains encrypted until the peer devices stay connected.
- Phase 3 (Bonding, Sharing and Storage of Permanent keys): Devices enter this phase if bonding is required. In this phase, permanent keys (Long Term Key or LTK) are exchanged between two devices using the encrypted link, which was established in phase 2 using temporary keys. These permanent keys are then stored in non-volatile memory to be made available for the devices over each connection.

BLE Framework Module Limitations

1. The BLE framework is tested only on the RL78G1D BLE hardware module. Support for

different BLE modules will be added in later versions.
2. BLE Framework using RL78G1D will see compilation warnings. All the warnings are in the 3rd party RL78G1D driver code. The BLE framework files do not have any warning. These warnings should not impact the user applications.
3. The custom profile support in the BLE framework is limited to the RL78G1D type BLE hardware module only.
4. HID profile client mode, not supported by the RL78G1D BLE hardware module. As a result, the BLE framework implementation of the HID profile will also not support the HID profile client mode. Applications using BLE framework for RL78G1D will not be able to use the HID profile in client mode.
5. Multiple slave BLE devices cannot be connected to the RL78G1D BLE module.

The BLE framework is only tested on the following boards:

- DK-S7G2 Version 3.1
- DK-S3A7 Version 2.0
- PK-S5D9 Version 1.0
- ADK-S3A3
- TB-S5D5 Version 0.5D
- TB-S3A6 Version 0.5D
- DK-S128 Version 0.5b
- DK-S124 Version 3.1

Refer to the most recent SSP Release Notes for module limitations.

4.1.11.4 Including the BLE Framework Module in an Application

This section describes how to include the BLE Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the BLE Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the BLE Framework module is g_sf_ble0. This name can be changed in the associated Properties window.)

BLE Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_ble0 RL78G1D BLE GAP and GATT on sf_ble_rl78g1d	Threads	**New Stack> Framework> Networking> BLE> RL78G1D BLE GAP and GATT on sf_ble_rl78g1d**
g_sf_ble_onboard_profile0 On-Board Profile on RL78G1D BLE Framework	Threads	**New Stack> Framework> Networking> BLE> On-Board Profile on RL78G1D BLE Framework**

When the BLE Framework module on sf_ble is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing

additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

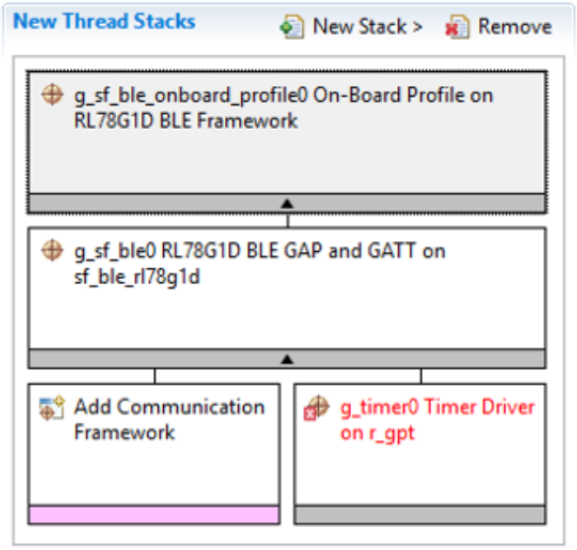


Figure 148: BLE Framework Module Stack

In the stack above, the Add Communication Framework block has not been populated yet. There are multiple possible selections for the Communication Frameowork; they are not all provided so as to not complicate needlessly, the figure and the following configuration tables. Typical options include:

- Communications Framework on sf_comms_telnet
- Communications Framework on sf_el_ux_comms_v2
- Communications Framework on sf_uart_comms

4.1.11.5 Configuring the BLE Framework Module

The BLE Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuring the On-Board Profile Profile on RL78G1D BLE Framework

Configuration Settings for the On-Board Profile on RL78G1D BLE Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Heart Rate Profile	Enabled, Disabled Default: Enabled	Heart rate profile selection.
Alert Notification Profile	Enabled, Disabled Default: Disabled	Alert notification profile selection.
Blood Pressure Profile	Enabled, Disabled Default: Disabled	Blood pressure profile selection.
Find Me Profile	Enabled, Disabled Default: Disabled	Find me profile selection.
HID Over GATT Profile	Enabled, Disabled Default: Disabled	HID gatt profile selection.
Health Thermometer Profile	Enabled, Disabled Default: Disabled	Health thermometer profile selection.
Phone Status Alert Profile	Enabled, Disabled Default: Disabled	Phone alert profile selection.
Proximity Profile	Enabled, Disabled Default: Disabled	Proximity profile selection.
Scan Parameter Profile	Enabled, Disabled Default: Disabled	Scan parameter profile selection.
Time Profile	Enabled, Disabled Default: Disabled	Time profile selection.
Name	g_sf_ble_onboard_profile0	Module name.
Name of generated initialization function	sf_ble_rl78g1d_onboard_profile_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the BLE GAP and GATT on sf_ble_rl78g1d

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_ble0	Module name.
Bluetooth Device Address (Restart Board after first run to see changed Address)	{0x0, 0x0, 0x0, 0x0, 0x0}	Bluetooth device address selection.
Address Type	Public Address, Random Address Default: Public Address	Address type selection.
Scan Interval	48	Scan interval selection.
Scan Window	48	Scan window selection.
Maximum Connection Interval	40	Maximum connection interval selection.
Connection Slave Latency	0	Connection slave latency selection.
Supervision Timeout	80	Supervision timeout selection.
BLE Driver Thread Priority	1	BLE Driver thread priority selection.
BLE Serial Thread Priority	1	BLE Serial thread priority selection.
Name of generated initialization function	sf_ble_rl78g1d_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuring the On-Board Profile Profile on RL78G1D BLE Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-

level modules.

Configuration Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR, called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest (Period-1) PCLK/Shortest: 1 PCLK, Longest: (Period-2) PCLK Default: Shortest 2 PCLK, Longest: (Period-1) PCLK	Select the duty cycle range. Due to hardware limitations, one PCLK is added before the output pin toggles after the duty cycle is reached. This extra clock cycle is added to the ON time (if Shortest: 2 PCLK is selected) or the OFF time (if Shortest: 1 PCLK is selected) based on this configuration.
Period Value	10	See Timer Period Calculation.
Period Unit	Milliseconds	See Timer Period Calculation.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts	Duty cycle unit selection.
Auto Start	True	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
GTIOCA Output Enabled	False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low	Controls output pin level when the timer is stopped.

GTIOCB Output Enabled	False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low	Controls output pin level when the timer is stopped.
Callback	RBLE_Timer_cb	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Overflow interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

BLE Framework Module Clock Configuration

The BLE Framework module uses the clocks specified by the lower-level modules.

BLE Framework Module Pin Configuration

The BLE Framework module uses the pins specified by the lower-level modules.

4.1.11.6 Using the BLE Framework Module in an Application

The steps in using the Bluetooth Low Energy Framework module in a typical application are:

1. Initialization- registration of callback functions, provisioning, and advertisement using the sf_ble_api_t::open API.

2. Provision the BLE module using the sf_ble_api_t::provisionSet API.

3. Scan for advertisement using the sf_ble_api_t::scan API.

4. Connect to remote BLE device using the sf_ble_api_t::connect API.

5. Enable the on-board profile using the sf_ble_onboard_profile_api_t::onbpEnable API.

6. Enable notification using the sf_ble_onboard_profile_api_t::onbpClientWriteCCCD API.

7. Write profile characteristics using the sf_ble_onboard_profile_api_t::onbpServerWriteData API.

8. Read profile characteristics using the sf_ble_onboard_profile_api_t::onbpClientReadChar API.

9. Disable the profile using the sf_ble_onboard_profile_api_t::onbpDisable API.

10. Disconnect from the remote BLE device using the sf_ble_api_t::disconnect API.

11. Close the BLE module using the sf_ble_api_t::close API.

The following figure illustrates common steps in a typical operational flow diagram:

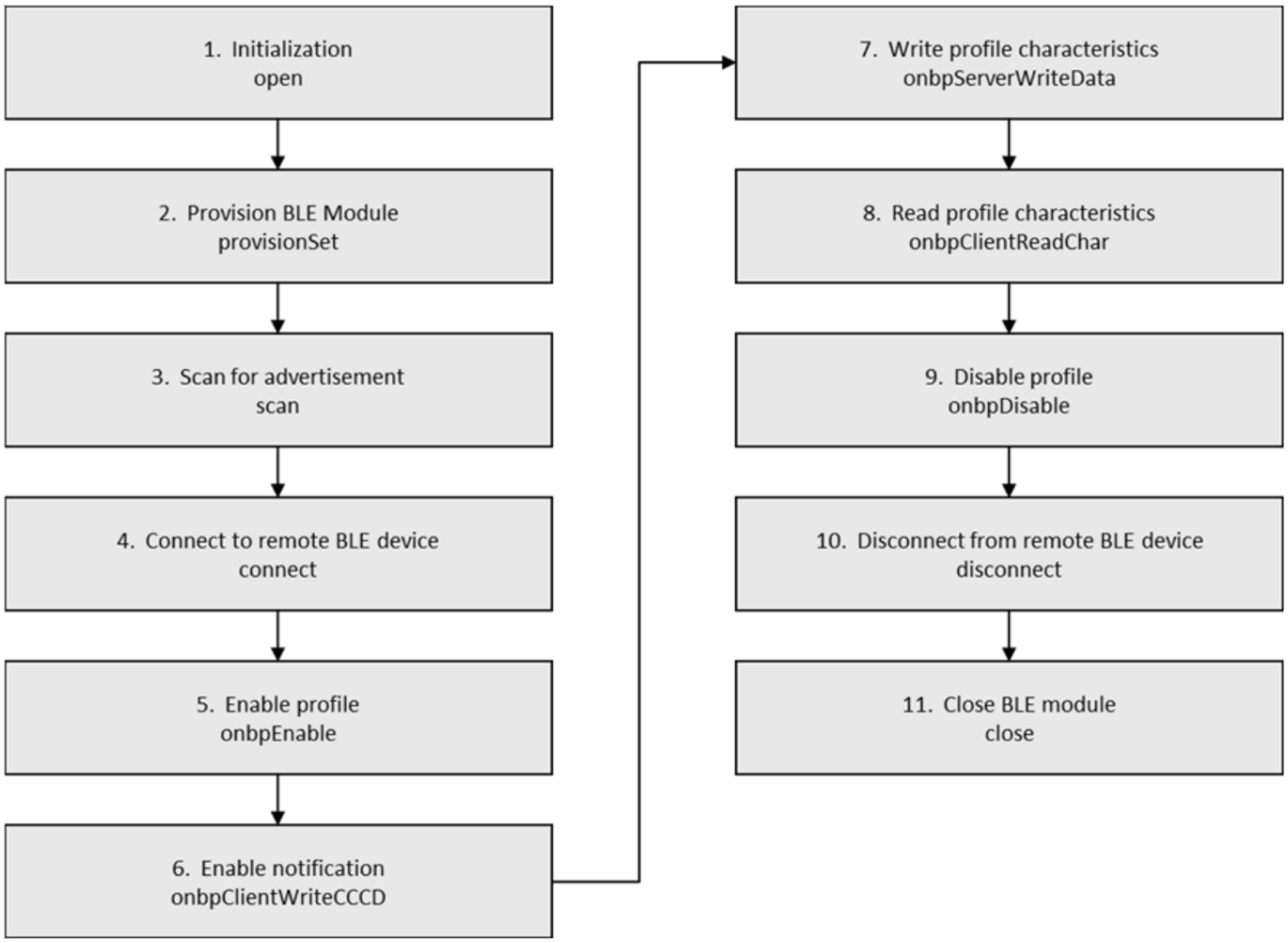


Figure 149: Application Control Flow using BLE Module Initialization

4.1.12 Cellular Framework

4.1.12.1 Cellular Framework Introduction

The Cellular Framework module provides a high-level application layer interface for cellular modem integration in SSP. The Cellular Framework provides a common interface for the applications to interface with the cellular modems from various vendors.

The Cellular Framework provides a set of APIs to provision, configure, and communicate with the cellular network for data communication. The Cellular Framework uses the UART Comms Framework to communicate with cellular modems over a serial interface by using AT commands. The Cellular framework creates the serial data pipe over a serial interface for the data communication, leveraging the PPP WAN protocol provided by NetX™. Data communication using TCP/IP can be established over this Wide Area Network (WAN) link using NetX Application protocols, sockets or IoT protocols such as MQTT.

The Cellular Framework also provides the framework-level socket APIs to communicate with the TCP/IP stack present on-chip (inside cellular hardware module) in certain cellular hardware modules and with the TCP/IP link for the network using socket APIs.

Cellular Framework Module Features

- Supports connectivity using:
  - BSD Socket interface for On-Chip stack present on the Cellular Module
  - NetX Stack on Synergy MCU (Host) using NSAL interface
- Supports a common a set of APIs to interface to the networking stack and a generic interface for the different Cellular hardware modules.
- Using generic APIs and abstraction, applications developed for the cellular hardware module can be easily migrated to work with another cellular hardware module.
- Supported Cellular modems:
  - RYZ014A CAT M1 (PMOD Expansion Board for RYZ014A) (For information on this kit, see: https://www.renesas.com/us/en/products/interface-connectivity/wireless-communications/cellular-iot-modules/rtkyz014a0b00000be-pmod-expansion-board-ryz014a)
  - Quectel BG96 (CAT M1, NB-IoT and GPRS) Rev F

Note

The following boards are obsolete and are no longer supported in SSP:

- NimbeLink CAT3 (NL-SW-LTE-TSVG, REVISION 17.01.571) Verizon-US
- NimbeLink CAT3 (NL-SW-LTE-TEUG, REVISION 17.01.571) India and Europe
- NimbeLink CAT1 (NL-SW-LTE-GELS3-B and NL-SW-LTE-GELS3-C, REVISION 4.3.3.0c) Verizon-US

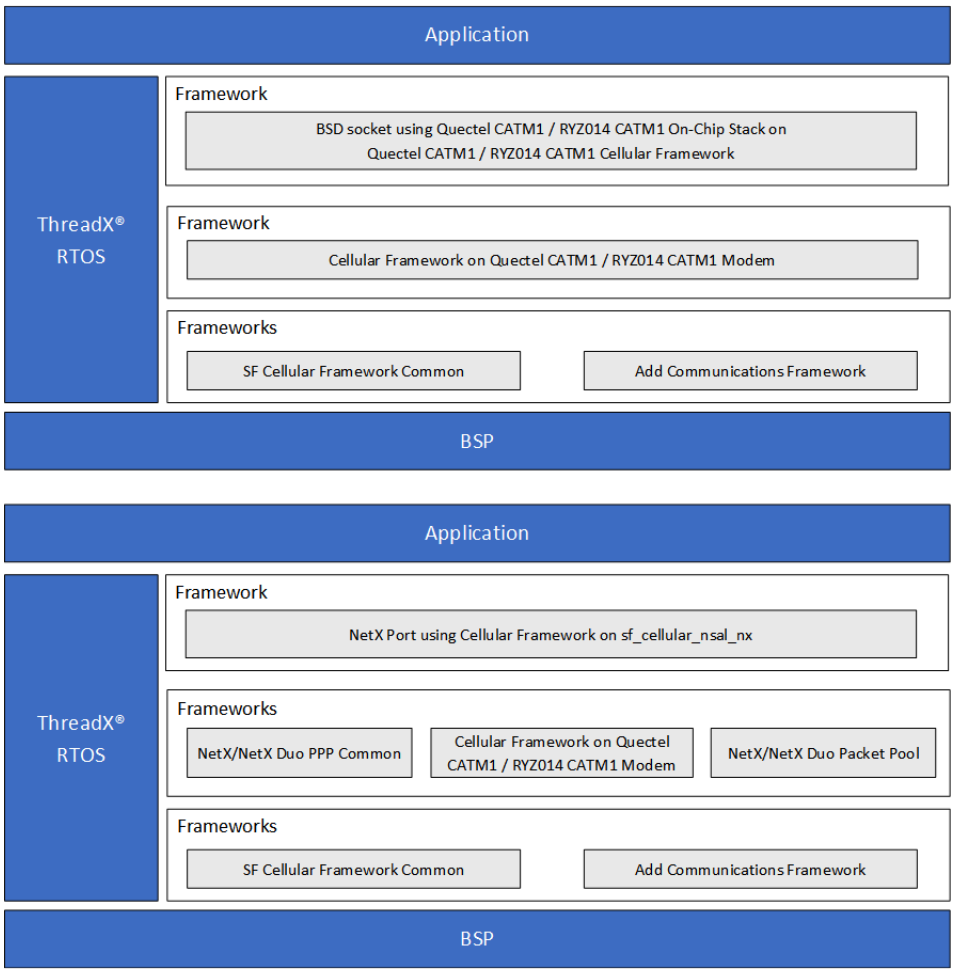


Figure 150: Cellular Framework Module Block Diagram

Note

Any of the four BSD Socket On-Chip stacks can implement any of the four Cellular Framework modules below it. Similarly, the NetX Port on sf_cellular_nsal_nx can implement any of the four Cellular Framework modules below it.

4.1.12.2 Cellular Framework Module APIs Overview

The Cellular Framework defines APIs for each of the related modules. The following descriptions explain the operation of each API.

Note

A more detailed description of the Cellular framework module APIs are available in the Cellular Application Note, downloadable from the Renesas web site. Just search, in the top search bar, for R30AN0311 and the application note and application project will be listed in the search results.

Cellular Framework Module APIs

Cellular Framework Module API Summary

Function Name	Example API Call and Description

open	g_sf_cellular0.p_api->open (g_sf_cellular0.p_ctrl, g_sf_cellular0.p_cfg); This API function initializes and enables the Cellular module. The open function returns the Cellular control structure, uniquely identifying the instance of the Cellular framework. The Cellular framework open function accepts the Cellular module configuration as an argument, with the following parameters: - Operator Selection Mode (enumeration) - Operator Name Format (enumeration) - Operator Name (string) - Preferred Operator List (array of structures) - Time zone update policy (enumeration)
close	g_sf_cellular0.p_api->close (g_sf_cellular0.p_ctrl); This API uninitializes the Cellular module and disables it. It takes the Cellular control structure as an argument.
infoGet	g_sf_cellular0.p_api->infoGet (g_sf_cellular0.p_ctrl, p_cellular_info); This API takes the Cellular control structure as an argument and returns the following information obtained from the Cellular module: - Chipset/driver information (string) - Manufacturer name (string) - Firmware version (string) - IMEI number (string) - RSSI value (unsigned integer 16 bits) - Bit Error Rate (unsigned integer 16 bits)
statisticsGet	g_sf_cellular0.p_api->statisticsGet (g_sf_cellular0.p_ctrl, p_stats); This API gets the data statistics from the Cellular module. It takes the Cellular control structure as an argument and returns the following statistics: - Received packets (unsigned integer 32 bits) - Transmitted packets (unsigned integer 32 bits) - Transmit packet errors (unsigned integer 32 bits)
transmit	g_sf_cellular0.p_api->transmit (g_sf_cellular0.p_ctrl, p_buffer, length); This API sends the data/packet out. It takes the Cellular control structure, the data buffer and the data buffer length as an argument. The Cellular framework transmit function passes the data buffer to the PPP driver for transmission.

provisioningGet	g_sf_cellular0.p_api->provisioningGet (g_sf_cellular0.p_ctrl, p_cellular_provisioninfo); This API takes the Cellular control structure as an argument and returns the following parameters: - Authentication type(enumeration) - Username (string) - Password (string) - APN Name(string) - PDP Context ID (integer) - PDP Context Type (enumeration) - Airplane mode (enumeration)
provisioningSet	g_sf_cellular0.p_api->provisioningSet (g_sf_cellular0.p_ctrl, p_cellular_provisioninfo); This API sets the authentication credential information. It takes the Cellular control structure and the following parameters as argument to provision the Cellular module: - Authentication type(enumeration) - Username (string) - Password (string) - APN Name(string) - PDP Context ID (integer) - PDP Context Type (enumeration) - Airplane mode (enumeration)
networkConnect	g_sf_cellular0.p_api-> networkConnect (g_sf_cellular0.p_ctrl); This API establishes the Network connection over Cellular Network, using which the application can communicate to remote host with the help of Network stack. It takes the Cellular control structure as an argument.
networkDisconnect	g_sf_cellular0.p_api->networkDisconnect (g_sf_cellular0.p_ctrl); This API terminates the Network connection established using networkConnect API. It takes the Cellular control structure as an argument.
simPinSet	g_sf_cellular0.p_api->simPinSet (g_sf_cellular0.p_ctrl, p_pin); This API allows the application/user to change the PIN required to register on Cellular Network. It takes the Cellular control structure, old PIN and New PIN as arguments.
simLock	g_sf_cellular0.p_api->simLock (g_sf_cellular0.p_ctrl, p_pin); This API locks the SIM. It takes the Cellular control structure and Sim PIN as arguments.

simUnlock	g_sf_cellular0.p_api->simUnlock (g_sf_cellular0.p_ctrl, p_pin); This API unlocks the SIM. It takes the Cellular control structure and Sim PIN as arguments.
simIDGet	g_sf_cellular0.p_api->simIDGet (g_sf_cellular0.p_ctrl, p_sim_id); This API reads the Sim ID from the Cellular module. It takes the Cellular control structure as argument and returns the SIM ID read from the Cellular module.
commandSend	g_sf_cellular0.p_api->commandSend (g_sf_cellular0.p_ctrl, p_input_at_commmand, p_output, timeout); Send AT command directly to Cellular Modem.
networkStatusGet	g_sf_cellular0.p_api-> networkStatusGet (g_sf_cellular0.p_ctrl, p_status); This API gets Cellular Module Network Status information. It takes the Cellular control structure as argument and returns following parameters: - Country code (integer) - Operator code (integer) - RSSI (integer) - Cell ID (string) - IMSI (string) - Operator name (string) - Service Domain (integer) - Active Band (integer).
versionGet	g_sf_cellular0.p_api->versionGet (p_version); This API retrieves the version for the API using the version pointer.
reset	g_sf_cellular0.p_api->reset (g_sf_cellular0.p_ctrl, reset_type); Reset the cellular hardware module.

Note

*** **When using commandSend API with RYZ014 cellular modem always give the exact size of the AT command in the p_input_at_commmand parameter.*
For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables review the SSP User's Manual API References for the associated module.

Cellular Framework Module Socket APIs Summary

These APIs can be used to configure the cellular module when using an on-chip networking stack, which helps to ping particular ip address, opening and closing of scokets and also to perform socket operations like TCP/UDP data transfer.

On-Chip Networking Stack Support Cellular Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_cellular_socket0.p_api->open (g_sf_cellular_socket0.p_ctrl, g_sf_cellular_socket0.p_cfg); This API calls the Cellular Framework's lower level open () API to Initialize the Cellular Device Driver.
close	g_sf_cellular_socket0.p_api->close (g_sf_cellular_socket0.p_ctrl); This API calls the Cellular Framework's lower level close () API to close the Cellular Device Driver.
versionGet	g_sf_cellular_socket0.p_api->versionGet (p_version); This API retrieves the version for the API using the version pointer.
ping	g_sf_cellular_socket0.p_api->ping (g_sf_cellular_socket0.p_ctrl, p_ip_address, count, interval_ms); This api pings the IP address provided by user.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

These API functions can be used by an application to perform data transfers using sockets. They include socket API functions which are compliant with these BSD API functions:

- socket
- close
- connect
- send
- recv
- sendto
- recvfrom
- select

Note

Ping functionality is supported only for RYZ014A CATM1 modem and is an unsupported feature for Quectel CATM1 modems.

Cellular Framework Error Codes

The following table lists the Cellular Framework specific error codes. These error codes are part of ssp_err_t.

Cellular Framework Error Codes

Error Codes	Description
SSP_SUCCESS	Call successful.

SSP_ERR_CELLULAR_CONFIG_FAILED	Configuration failed.
SSP_ERR_CELLULAR_INIT_FAILED	Initialization failed.
SSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmit failed.
SSP_ERR_CELLULAR_FW_UPTODATE	Up to date.
SSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Upgrade failed.
SSP_ERR_CELLULAR_FAILED	General failure.
SSP_ERR_CELLULAR_INVALID_STATE	Invalid state.
SSP_ERR_CELLULAR_REGISTRATION_FAILED	Registration failure.

4.1.12.3 Cellular Framework Module Operational Overview

The Cellular Framework provides a generic interface for applications to seamlessly communicate with the cellular hardware module from various vendors without the necessity of changing the applications. The framework mainly consists of a common set of APIs to interface to the networking stack and different cellular hardware modules. This section introduces the Cellular Framework's basic blocks and key features that enable you to determine whether the intended cellular application can be developed using the Cellular Framework.

Note

Additional operational descriptions of the Cellular Framework module are available in the Cellular Application Note, downloadable from the Renesas web site. Search (in the search bar at the top of the screen) for R30AN0311 and the application note and application project will be listed in the search results.

With the provided API and abstraction, the applications developed for the cellular hardware module can be easily ported to use another cellular hardware module. The networking stack NetX is also integrated with the framework using the Network Software Abstraction Layer (NSAL).

The Synergy Cellular Framework consists of the following logical blocks:

- Synergy Cellular Framework Application Interface.
- Network Stack Abstraction Layer (NSAL) for NetX TCP/IP stack.
- Cellular Device Driver (AT command interface for interacting with the cellular chipset).
- BSD Socket compatible APIs for interfacing with Cellular hardware module that supports on-chip networking stack.
- Synergy Software Package(SSP) HAL Interface.

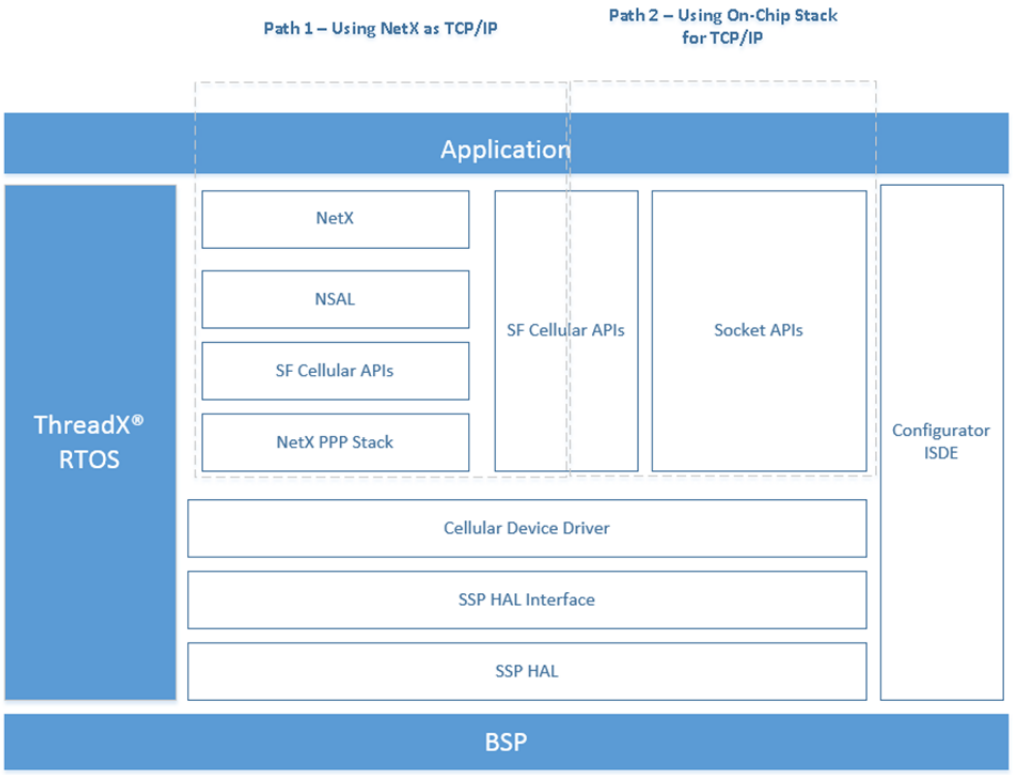


Figure 151: Cellular Framework Module Application Perspective

The Cellular Framework provides a common set of interfaces for the application to configure, provision and to communicate with the cellular hardware module. By using these generic interfaces, the user can develop a cellular-based application using Synergy MCUs. The cellular hardware module has various configuration parameters as specified by the family of 3GPP standards. It is possible that individual device drivers and/or cellular chipsets/modules will not support configuration of all parameters. At a bare minimum, the network operator, Access Point Name (APN) and security credentials are required to make the module functional.

Network Stack Abstraction Layer

The Cellular Framework provides a network stack abstraction layer (NSAL). The NSAL is layer that connects the NetX and the cellular driver by using a (PPP) stack that is used for data communication over a WAN link.

Socket Interface Layer

The Cellular Framework provides a socket-level API for the application to interact with the on-chip networking stack present on the cellular hardware module. This requires the cellular hardware module/driver to support an on-chip networking stack and socket interface. When the application uses these APIs, it uses the on-chip networking stack present on the cellular hardware module and does not use the NSAL or the NetX and its socket APIs and does not use the Networking stack running on the Synergy MCU Group.

PPP Stack

Point to point protocol (PPP) is a widely used WAN protocol in data communication. The PPP stack is a part of the NetX stack available in the SSP. The NSAL leverages the PPP stack to communicate over

the serial interface to the cellular service provider's network. PPP configuration provides options for authentication methods like PAP/CHAP; these authentication mechanisms are optional for the link establishment. The NSAL makes use of framework APIs to send/receive data from the cellular hardware module. The NSAL also allows the cellular device driver to be re-used without any specific changes to the network stack.

Cellular Device Driver (AT Command Interface For Interacting With The Cellular Chipset)

The Cellular Framework uses the AT command set to interact with the cellular modem using the serial driver. The serial interface used to interact with the modem is the UART. The UART speed used in the framework defaults ranges up to 115200 bits/sec.

Cellular Framework Module Important Operational Notes and Limitations

Cellular Framework Module Operational Notes

The application can be used in two different paths for the communication using the framework, depending on the support available on the cellular modems. Some modules provide options to use the TCP/IP stack at the host end and other modules provide options to use the TCP/IP stack present on the cellular modem itself. In some cases, the cellular hardware module provides both. When the host TCP/IP stack (NetX) is used, the logical layers of NetX, the NSAL and the PPP are used as described in the architecture diagram. When the on-chip stack is used, the socket APIs are used to communicate with the TCP/IP stack present on the cellular modem. However, the user cannot use both at the same time.

In the sf_cellular_qctlcatm1 framework, AT commands should be passed after the "APP RDY" string. However, in the current sf_cellular_qctlcatm1 framework, AT commands are being passed before the "APP RDY" string is received. There is no functional impact because, in the current sf_cellular_qctlcatm1 framework, instead of checking the "APP RDY" string, "OK" response is being checked in the BG 96 modem, which ensures that the modem is working.

Cellular Framework Module Initialization

As shown in the following control flow diagram, during the initialization using the configuration supplied by the user as required for the cellular modem, nx_ip_create is called. This API internally invokes the NSAL driver entry function that takes care of the link-level initialization and initializes the cellular hardware module. Additionally, it provisions the module and establishes the Network connection using the PPP interface.

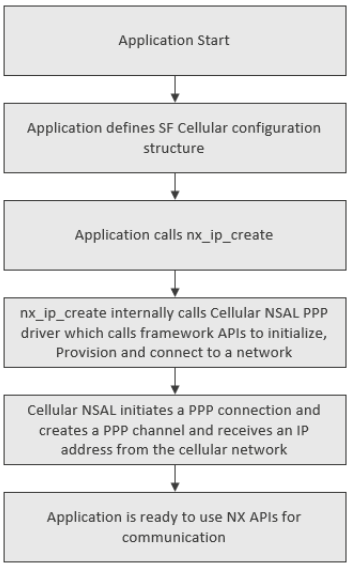


Figure 152: Cellular Framework Module Initialization Sequence

Cellular Hardware Module Provisioning

During the provisioning of the cellular modem, control structure and user configuration structures are passed as arguments. The details of the user arguments used for provisioning are the authentication, APN, username and password.

Application Flow Control Using the Socket Interface

The following flow diagram shows the flow for the on-chip stack path usage with the Cellular Socket interface.

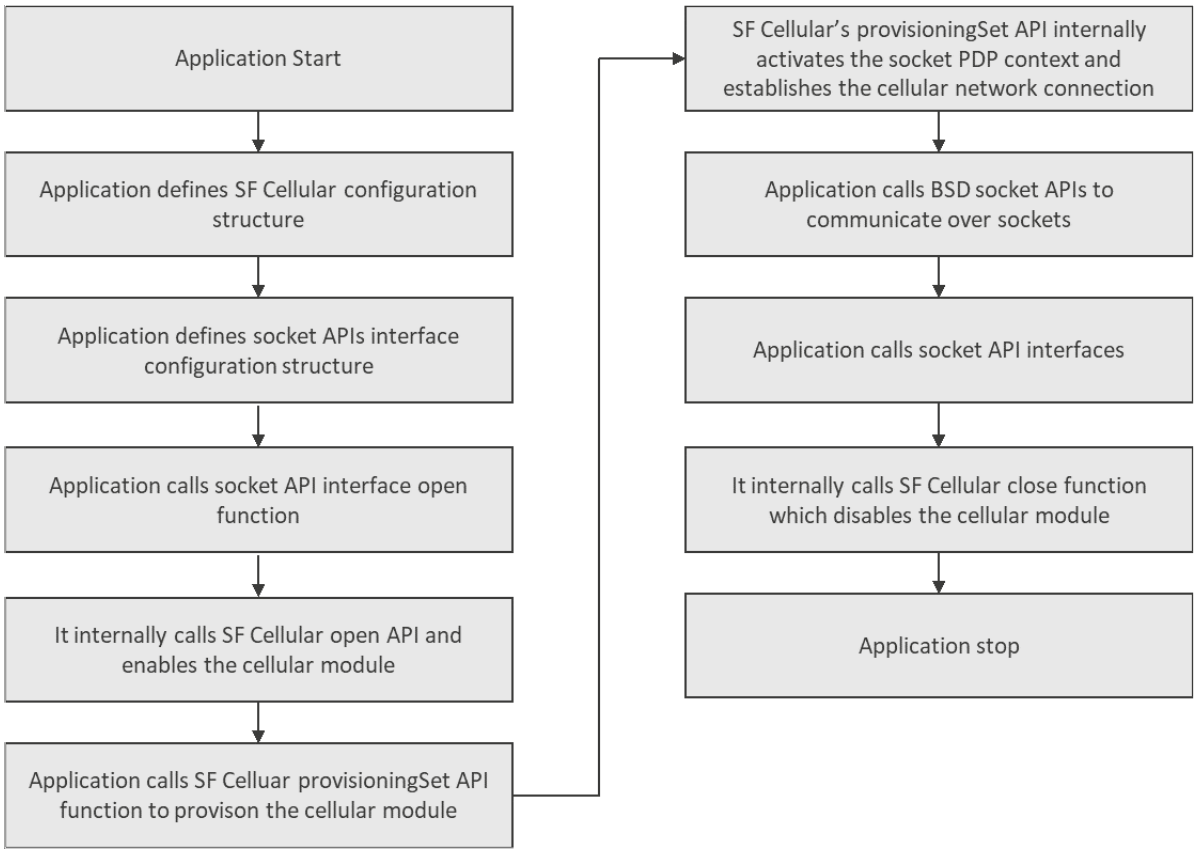


Figure 153: Cellular Framework Module Socket Interface

Cellular Packet Transmission

The following flow diagram shows the sequence of steps that the packet transmission uses for the NetX application.

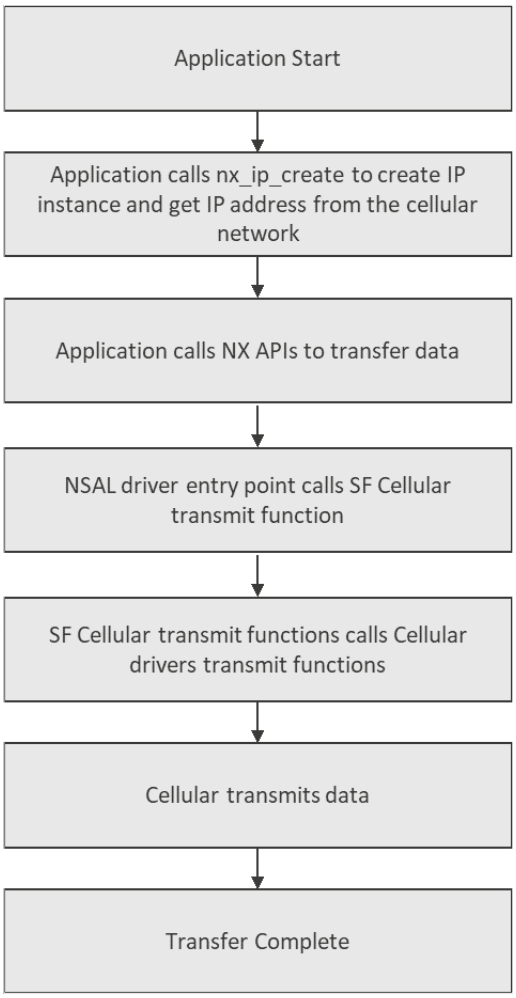


Figure 154: Cellular Framework Module Packet Transmission Sequence

Cellular Packet Reception

The following flow diagram shows the Packet reception for the Cellular Framework using NetX. In the case of a receive, when the data is received on the serial interface, the processing thread triggers the callback function and the callback functions handles the data and sends it to the NetX layers for further processing.

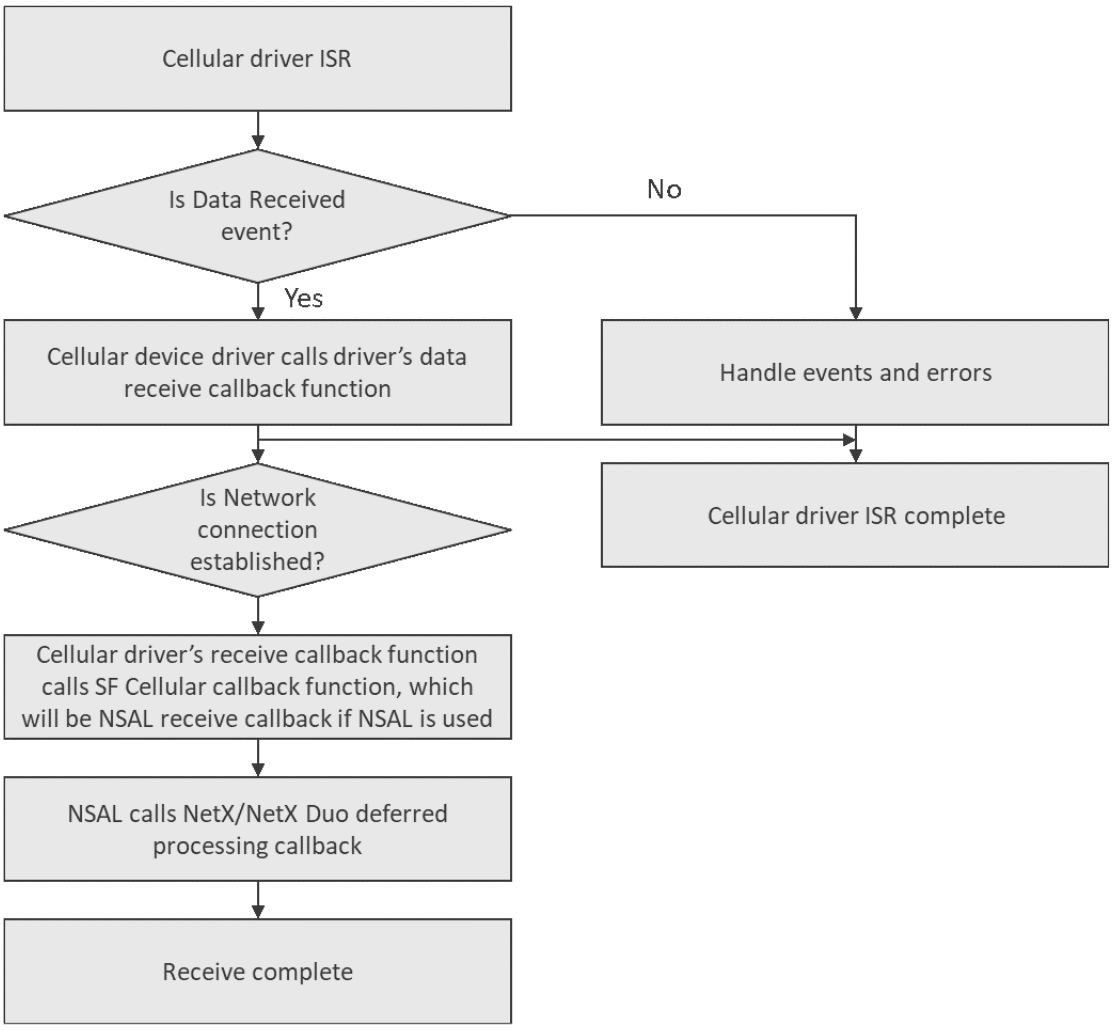


Figure 155: Cellular Framework Module Packet Reception Sequence

Cellular Module baud rate

The Cellular Framework baud rate update feature works in the following stages:

- Check if the modem is operating at the baud rate set by the user in ISDE. If so, proceed with modem initialization.
- If the modem is not responding over the user specified baud rate, auto detect the baud rate at which the modem is currently operating.
- Switch the modem to the baud rate configured by the user in the ISDE. (This baud rate is then saved on the module).

The developer can configure the Baud rate of the UART under SF_CELLULAR in ISDE configuration as shown in the below image. This is the baud rate at which user wants the Modem to operate.

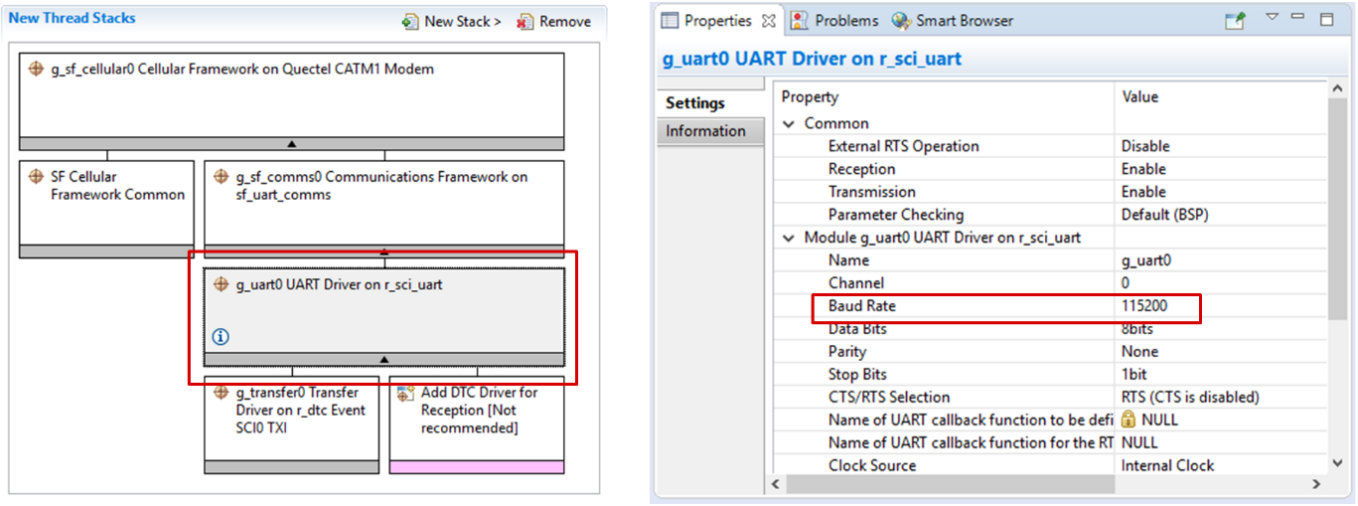


Figure 156: ISDE configuration for baud rate

Operational Steps

1. The Cellular Framework will try to communicate over the baud rate specified in the ISDE. If the module responds over that Baud rate, then the baud rate change feature will not be initiated.
2. If the Modem does not respond to the ISDE configured baud rate, then the Cellular Framework will detect the baud rate at which the Modem is operating currently. Once the baud rate is detected, the Cellular Framework will change the baud rate of the Modem to the user configured baud rate. The Framework will then save the baud rate configuration in the Modem.

    - *Note*

      The Cellular Framework will detect the baud rate of the Modem from the following list of baud rates {115200, 9600, 921600, 4800, 14400, 19200, 38400, 57600, 230400, 460800}. The baud rate configured in the ISDE will be skipped from the above list.
3. Once the Baud rate of the Modem is configured, the Cellular Framework will proceed with Modem initialization.

Framework close sequence

When using the Cellular framework with NSAL, that is, with NetX/NetX Duo, the application should call the NetX nx_ip_delete() API directly.

When using the On chip networking stack, the application shouldcall the sf_cellular_api_t::close API from the BSD socket interface, which internally calls the Cellular framework module sf_cellular_api_t::close API.

Runtime configuration support for SIM Properties

The cellular framework provides provision to set the SIM Pin and PUK value of SIM as a ISDE XML properties to unlock the SIM Card. These properties are used by SF Cellular framework to unblock the SIM access if SIM is in Lock state. In addition to these properties, Cellular framework also provides a callback function property to read the SIM Pin and PUK value at runtime. If user sets this callback function, and Cellular framework founds that SIM is in Locked state (either SIM Pin or SIM PUK Lock) then Cellular framework calls the callback function with the SIM Lock status. Based on the Sim Lock status, user should provide SIM Pin and/or SIM PUK information to callback function as mentioned

below:

1. If SIM Status is SF_CELLULAR_SIM_STATUS_PIN_REQUIRED, user should provide SIM Pin information.
2. If SIM Status is SF_CELLULAR_SIM_STATUS_PIN_PUK_REQUIRED then user should provide SIM Pin and PUK information in callback function.

Below is an illustrative code snippet for the callback function showing the configuration of the SIM Pin/PUK information at runtime.

```
uint8_t * const p_sim_pin = (uint8_t *)"1111";
uint8_t * const p_sim_puk = (uint8_t *)"12341234";

/** Sim Pin/PUK Read callback */
ssp_err_t sf_cellular_read_sim_pin_info_callback(sf_cellular_sim_pin_info_t  * p_args)
{
   if (p_args)
   {
      /** SIM Information found */
      switch(p_args->sim_status)
      {
         case SF_CELLULAR_SIM_STATUS_PIN_REQUIRED:
         p_args->p_sim_pin = (uint8_t *) p_sim_pin;
         break;

         case SF_CELLULAR_SIM_STATUS_PIN_PUK_REQUIRED:
         p_args->p_sim_puk = (uint8_t *) p_sim_puk;
         p_args->p_sim_pin = (uint8_t *) p_sim_pin;
         break;

         default:
         break;
      }
   }

   return SSP_SUCCESS;
}
```

Note

If SIM is in SIM Lock state and user entered invalid SIM Pin for 3 times, SIM get locked to SIM PUK state. SIM PUK pin is needed to collect from Cellular network providers. Entering invalid SIM PUK can block SIM permanently.

Cellular Framework Module Limitations

- The current framework supports the following Cellular modules:
  - RYZ014 CATM1 (RYZ014A PMOD).
  - Quectel BG96 (CAT M1, NB-IoT and GPRS) with firmware version BG96MAR02A07M1G_01.008.01.008
- RYZ014 does not support
  - NBIOT network band.
  - GSM network and GSM fallback sequence.
- The next section, Updating the Quectel BG96 firmware, includes instructions for downloading and updating the firmware for BG96 on the modules. Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Updating the Quectel BG96 Firmware

To determine the current version of firmware on the module, issue the following commands:

- ATI

- AT+QGMR

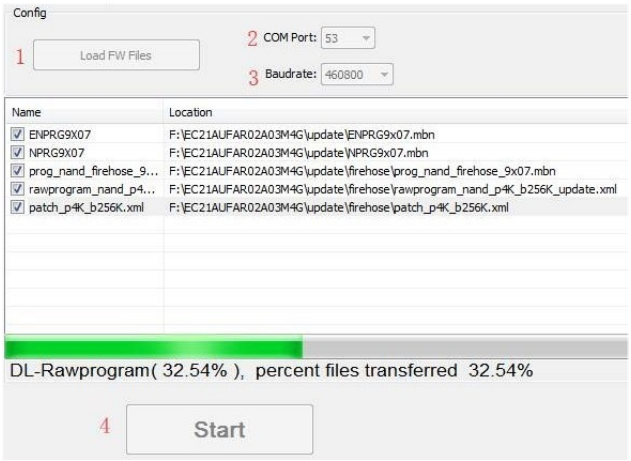
To update (flash) the firmware for the Quectel BG96 (CAT M1, NB-IoT and GPRS), follow these instructions:

1. First, download the QFLASH firmware update utility from the Quectel BG96 product download page: https://www.quectel.com/ProductDownload/BG96.html. Click the Download button and open the zip file.
2. From the Tool folder, extract the QFLASH tool.
3. From the Driver folder, extract the Quectel LTE Windows USB Driverand follow the steps below to install the LTE Driver:

    a. Unzip and install the LTE Driver on your PC, then connect the USB port of the module to the PC with a USB cable. Check if the DM port of USB appears in device manager:

4. Download the firmware from https://support.quectel.com. For information on this process, see the Renesas Knowledge Base: https://en-support.renesas.com/knowledgeBase/18243404.
5. Upzip firmware and QFLASH file and follow these steps:

    a. Click Load FW Files and load the firmware path (shown as 1 in the figure below):



    b. Choose DM port (shown as 2 in the figure above).
    c. Choose 460800 baud rate (shown as 3 in the figure above).
    d. Click Start.
    e. Normally, it takes a while for the upgrade to complete and display the success message. Check module firmware with "ATI" command.

4.1.12.4 Including the Cellular Framework Module in an Application

This section describes how to include the Cellular Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Cellular Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Cellular Framework module is g_sf_cellular0. This name can be changed in the associated Properties window.)

Cellular Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_cellular_socket0 BSD Socket using RYZ014 CATM1 / Quectel CATM1 On-Chip Stack on RYZ014 CATM1 / Quectel CATM1 Cellular Framework	Threads	New Stack> Framework> Networking> Cellular> BSD Socket using RYZ014 CATM1 / Quectel CATM1 On-Chip Stack on RYZ014 CATM1 / Quectel CATM1 Cellular Framework
g_sf_cellular_0 Cellular Framework on RYZ014 CATM1 / Quectel CATM1 Modem	Threads	New Stack> Framework> Networking> Cellular> Cellular Framework on RYZ014 CATM1 /Quectel CATM1 Modem
g_sf_el_nx0 NetX Port using Cellular Framework on sf_cellular_nsal_nx	Threads	New Stack> Framework> Networking> Cellular> NetX Port using Cellular Framework on sf_cellular_nsal_nx

When the Cellular Framework module on sf_cellular is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

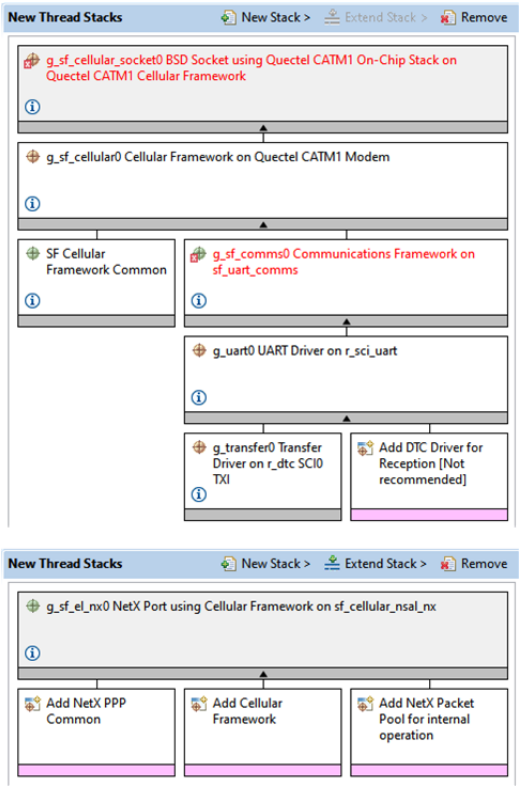


Figure 157: Cellular Framework Module Stack

In the stack above, the Add SF Communications Framework block has not been populated yet. There are multiple possible selections for the Communication Framework; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. Communications Framework on sf_uart_comms should be used when using the sf_cellular framework.

4.1.12.5 Configuring the Cellular Framework Module

The Cellular Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the BSD Socket Using RYZ014 CATM1 On-Chip Stack on RYZ014 CATM1 Cellular Framework

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_cellular_socket0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the BSD Socket Using Quectel CATM1 On-Chip Stack on Quectel CATM1 Cellular Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_cellular_socket0	Module name.
Name of generated initialization function	sf_cellular_qctcatm1_socket_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Cellular Framework on Quectel CATM1 Modem

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection.
AT Command Retry Count	5	Modem selection.
Name	g_sf_cellular0	Module name.
SIM Pin (Used to Unlock SIM)	1111	SIM Pin selection.
SIM PUK Pin (Used to Unlock SIM)	12345678	SIM PUK Pin selection.
SIM Pin/PUK Callback to read SIM Pin/PUK value at runtimeNULL	NULL	SIM Pin/PUK read callback selection.
Number of Preferred Operator	0	Number of preferred operator selection.
Preferred Operator 1 Name	40422	Preferred operator 1 name selection.

Preferred Operator 1 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 1 name format selection.
Preferred Operator 2 Name	40424	Preferred operator 2 name selection.
Preferred Operator 2 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 2 name format selection.
Preferred Operator 3 Name	40422	Preferred operator 3 name selection.
Preferred Operator 3 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 3 name format selection.
Preferred Operator 4 Name	40424	Preferred operator 4 name selection.
Preferred Operator 4 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 4 name format selection.
Preferred Operator 5 Name	40422	Preferred operator 5 name selection.
Preferred Operator 5 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 5 name format selection.
Operator Select Mode	Auto, Manual, Deregister, Manual Fallback Default: Auto	Operator select mode selection.
Operator Name (Manual Mode Selection)	40422	Operator name selection.
Operator Name Format (Manual Mode Selection)	Long, Short, Numeric Default: Numeric	Operator name format selection.
Time Zone Update Policy	Enabled, Disabled Default: Enabled	Time zone update policy selection.
Receive Data Callback	sf_cellular_nsal_recv_callback	Receive data callback selection.
Provisioning Callback	celr_prov_callback	Provisioning callback selection.
Circular Queue Size in Bytes	256	Circular queue size selection.
SF Communications Framework Thread Stack Size	512	SF communications framework thread stack size selection.
Numerical priority of SF Communication Framework Thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.	5	Numerical priority of SF communication framework thread selection.
Cellular Module Reset IO Pin	IOPORT_PORT_01_PIN_06	Cellular module reset IO pin selection.

Network Scan Sequence	LTE cat.M1-> LTE Cat.NB1-> GSM, LTE Cat.M1-> GSM-> LTE Cat.NB1, GSM-> LTE Cat.NB1-> LTE Cat.M1, GSM-> LTE Cat.M1-> LTE Cat.NB1, LTE Cat.NB1 -> LTE Cat.M1 -> GSM, LTE Cat.NB1 -> GSM -> LTE Cat.M1 Default: LTE cat.M1-> LTE Cat.NB1-> GSM	Network scan sequence selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Cellular Framework on RYZ014 CATM1 Modem

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection.
AT Command Retry Count	5	Modem selection.
Name	g_sf_cellular0	Module name.
SIM Pin (Used to Unlock SIM)	1111	SIM Pin selection.
SIM PUK Pin (Used to Unlock SIM)	12345678	SIM PUK Pin selection.
SIM Pin/PUK Callback to read SIM Pin/PUK value at runtimeNULL	NULL	SIM Pin/PUK read callback selection.
Number of Preferred Operator	0	Number of preferred operator selection.
Preferred Operator 1 Name	40422	Preferred operator 1 name selection.
Preferred Operator 1 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 1 name format selection.
Preferred Operator 2 Name	40424	Preferred operator 2 name selection.
Preferred Operator 2 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 2 name format selection.
Preferred Operator 3 Name	40422	Preferred operator 3 name selection.
Preferred Operator 3 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 3 name format selection.

Preferred Operator 4 Name	40424	Preferred operator 4 name selection.
Preferred Operator 4 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 4 name format selection.
Preferred Operator 5 Name	40422	Preferred operator 5 name selection.
Preferred Operator 5 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 5 name format selection.
Operator Select Mode	Auto, Manual Default: Auto	Operator select mode selection.
Operator Name (Manual Mode Selection)	40422	Operator name selection.
Operator Name Format (Manual Mode Selection)	Long, Short, Numeric Default: Numeric	Operator name format selection.
Time Zone Update Policy	Enabled, Disabled Default: Enabled	Time zone update policy selection.
Receive Data Callback	sf_cellular_nsal_recv_callback	Receive data callback selection.
Provisioning Callback	celr_prov_callback	Provisioning callback selection.
Circular Queue Size in Bytes	256	Circular queue size selection.
SF Communications Framework Thread Stack Size	512	SF communications framework thread stack size selection.
Numerical priority of SF Communication Framework Thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.	5	Numerical priority of SF communication framework thread selection.
Cellular Module Reset IO Pin	IOPORT_PORT_04_PIN_14	Cellular module reset IO pin selection.

Note

To use BSD Socket over RYZCATM1 on-chip stack, the "On-Chip Stack Support" property must be enabled. The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Cellular Framework Common Instance

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuring the NetX Port using the Cellular Framework

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules.

Configuration Settings for the NetX Port using the Cellular Framework on sf_cellular_nsal_nx

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_el_nx0	Module name.
PPP Stack Size in Bytes	2048	PPP stack size selection.
Name	g_nx_ppp0	Module name.
Numerical priority of PPP Thread	3	Specify the PPP thread priority. The priority must be lower than IP Helper thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.
Authentication Method	None, PAP, CHAP Default: None	Authentication method selection.
Invalid Packet Handler Callback	NULL	Invalid packet handler callback selection.
Link Down Callback	ppp_link_down_callback	Link down callback selection.
Link Up Callback	ppp_link_up_callback	Link up callback selection.
PAP Login Callback	NULL	A user callback function can be provided.
PAP Verify Login Callback	NULL	A user callback function can be provided.
Get Challenges Values Callback	NULL	A user callback function can be provided.
Get Responder Values Callback	NULL	A user callback function can be provided.
Get Verification Callback	NULL	A user callback function can be provided.
Local IPv4 Address (use commas for separation)	0,0,0,0	Local IPv4 address selection.
Peer IPv4 Address (use commas for separation)	0,0,0,0	Peer IPv4 address selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX PPP Common Instance

ISDE Property	Value	Description
Name	g_nx_ppp_common0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size (bytes)	1568	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Common on nx Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration changes for file download over cellular framework

HTTP file download of up to 2 MB is tested on BG96. Following is the list of ISDE properties that must be updated for downloading of files over HTTP while using Cellular framework.

ISDE configuration parameters for HTTP download using BG96 RevF module

ISDE Property	Value	Description
Packet Size of NetX packet pool instance (bytes)	1568 Default: 1568	Packet size selection.
Number of Packets in Pool for NetX packet pool instance	64 Default:16	Number of packets in pool selection.
IP thread priority	2 Default: 3	Thread priority of IP helper thread for NetX IP instance.
Numerical priority of SF Communication Framework thread	3 Default: 5	Priority value of SF Comms thread in g_sf_cellular block.
TCP socket Window Size	16384 (for GSM network 8192) Default: 1024	TCP socket window size of g_http_client.

ISDE configuration parameters for HTTP download using RYZ014 module

ISDE Property	Value	Description
Packet Size of NetX packet pool instance (bytes)	1568 Default: 1568	Packet size selection.
Number of Packets in Pool for NetX packet pool instance	64 Default:16	Number of packets in pool selection.
IP thread priority	2 Default: 3	Thread priority of IP helper thread for NetX IP instance.
Numerical priority of SF Communication Framework thread	3 Default: 5	Priority value of SF Comms thread in g_sf_cellular block.
TCP socket Window Size	16384 Default: 1024	TCP socket window size of g_http_client.

Cellular Framework Module Clock Configuration

The Cellular Framework module uses the clocks required for the specific selections of the lower-level modules.

Cellular Framework Module Pin Configuration

The Cellular Framework module uses input and output pins depending on the selections of the lower-level modules.

4.1.12.6 Using the Cellular Framework Module in an Application

In a typical Cellular application, much of the work is done by SSP based on the configured cellular module stack. When the IP instance along with the Cellular framework is added using the configurator, it includes the PPP stack as part of the framework. In addition, it also includes the NSAL and cellular device driver code. The auto-generated code is responsible for cellular initialization.

The user added code is responsible for the data connections and the ICMP ping. It is responsible for

sending the ping request to the user entered Public IP address and for verifying the ping response. Callback functions can be implemented for PPP link down, PPP link up and cellular provisioning.

The steps in using the Cellular Framework in a typical application are:

1. Initialization from generated code.
2. Wait for link to come up using the nx_ip_status_check API.
3. Ping the network using the nx_icmp_ping API.
4. Check for Event flag using the nx_event_flags_set API.

The following figure illustrates common steps in a typical operational flow diagram:

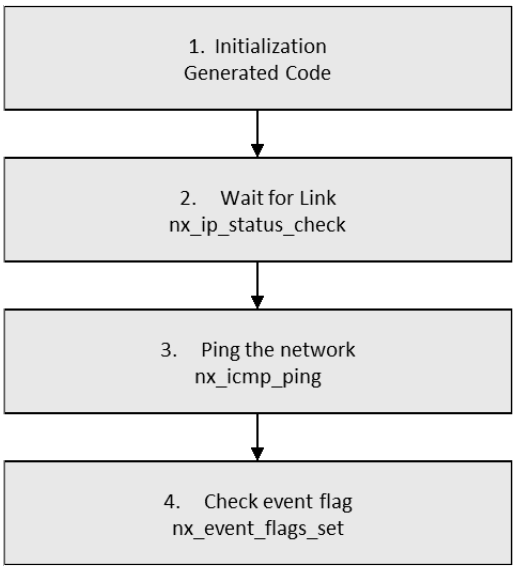


Figure 158: Application Control Flow using Cellular Module Initialization

4.1.13 Telnet Communications Framework on sf_comms_telnet

4.1.13.1 Telnet Communications Framework Introduction

The Communications Framework on NX provides a high-level API for communications framework applications and uses the Ethernet peripheral on the Synergy MCU.

Telnet Communications Framework Module Features

- High-level connectivity is supported on Ethernet but is easily changeable to UART and USB connectivity without API modification
- Supports channel locking for exclusive access
- Thread-aware implementation uses mutex and event flags internally

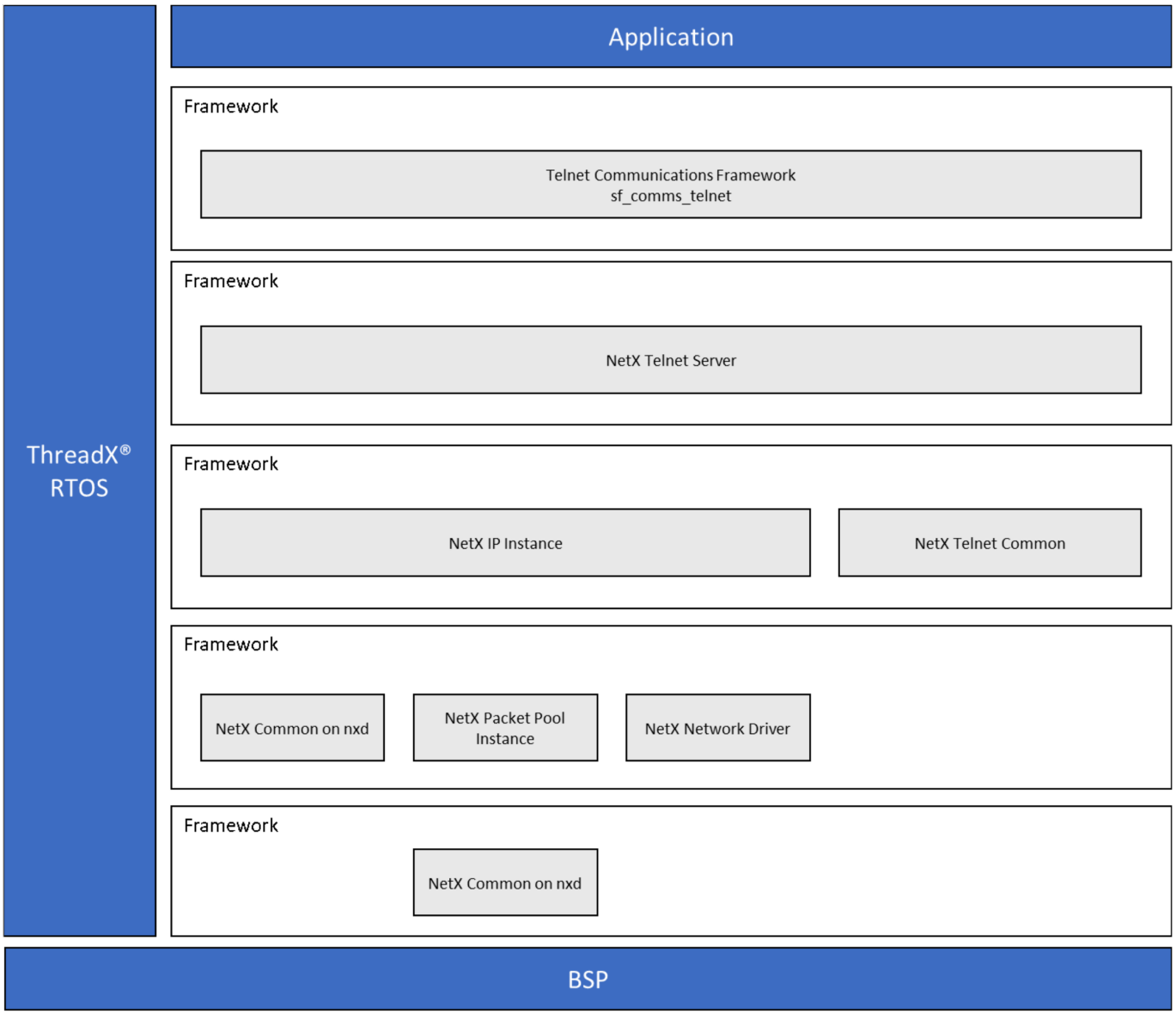


Figure 159: Telnet Communications Framework Module Block Diagram

4.1.13.2 Telnet Communications Framework Module APIs Overview

The Telnet Communications Framework module defines APIs to open, read from, write to, lock, unlock and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Telnet Communications Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_comms_telnet0.p_api->open (g_sf_comms_telnet0.p_ctrl, g_sf_comms_telnet0.p_cfg); Initialize the module.

read	g_sf_comms_telnet0.p_api->read (g_sf_comms_telnet0.p_ctrl, p_dest, bytes, timeout); Read a number of bytes of data into the destination.
write	g_ sf_comms_telnet0.p_api->write (g_sf_comms_telnet0.p_ctrl, P_src, bytes, timeout); Write a number of bytes from the source.
lock	g_sf_comms_telnet0.p_api->lock (g_sf_comms_telnet0.p_ctrl, locktype, timeout); Acquire lock type for the Telnet comms instance.
unlock	g_sf_comms_telnet0.p_api->unlock (g_sf_comms_telnet0.p_ctrl, locktype); Release the lock type for the Telnet comms instance.
close	g_sf_comms_telnet0.p_api->close (g_sf_comms_telnet0.p_ctrl); Disconnect Telnet server and clean up resources.
versionGet	g_ sf_comms_telnet0.p_api->versionGet(&version); Get version and store it in provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	A parameter is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.
SSP_ERR_UNSUPPORTED	Command not supported.
SSP_ERR_OUT_OF_MEMORY	Cannot allocate pool memory.
SSP_ERR_TIMEOUT	An event timed out.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.13.3 Telnet Communications Framework Module Operational Overview

The Communications Framework using Telnet on NX provides an easy to use connection over an Ethernet port. The high-level APIs are compatible with other connection protocols, such as UART and USB, so that it is easy to switch from one implementation to another without changing APIs.

Operations supported by the framework include initializing the module using the sf_comms_api_t::open API, and closing the module using the sf_comms_api_t::close API. A communications read is implemented by the sf_comms_api_t::read API and a communications write by the sf_comms_api_t::write API. The sf_comms_api_t::read and sf_comms_api_t::write lock the module only until the called API is in action.

During a read operation, when using the TX_WAIT_FOREVER timeout, if the Ethernet link goes down, the sf_comms_api_t::read API will continue to wait for the data from the read queue. Once the Ethernet link is back up, the read operation resumes and exits. To exit the read operation during a link down event, the user must explicitly abort the read operation by calling the sf_comms_api_t::close API in the link status change callback function. The sf_comms_api_t::lock API locks the module until the sf_comms_api_t::unlock API is called on the same module instance. This helps ensure that processing is completed before moving to the next API function call.

A user defined disconnect callback function (Telnet client disconnect callback) can be defined by the user to handle the Telnet client disconnection due to Client inactivity timeout. This callback will be called when the configured Client inactivity timeout occurs.

The underlying NetX driver supports the configuration of the IP address, the Network mask, and the Ethernet channel. When a different communications implementation is used (like USB) different low-level module configuration settings are used to define its interface. Thus, no code needs to be changed in the application, only need changes to configuration settings. The same API calls are retained at the application level.

Telnet Communications Framework Module Important Operational Notes and Limitations

Telnet Communications Framework Module Operational Notes

- The Ethernet peripheral can use either RMII or MII depending on MCU capabilities.

Telnet Communications Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.13.4 Including the Telnet Communications Framework Module in an Application

This section describes how to include the Telnet Communications Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Telnet Communications Framework module to an application, simply add it to a HAL

/Common thread using the stacks selection sequence given in the following table. (The default name for the Telnet Communications Framework module is sf_comms_telnet0. This name can be changed in the associated Properties window.)

Telnet Communications Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
sf_comms_telnet0 Telnet Communications Framework on sf_comms_telnet	Threads	**New Stack> Framework> Connectivity> Telnet Communications Framework on sf_comms_telnet**

When the Telnet Communications Framework module on sf_comms_telnet is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

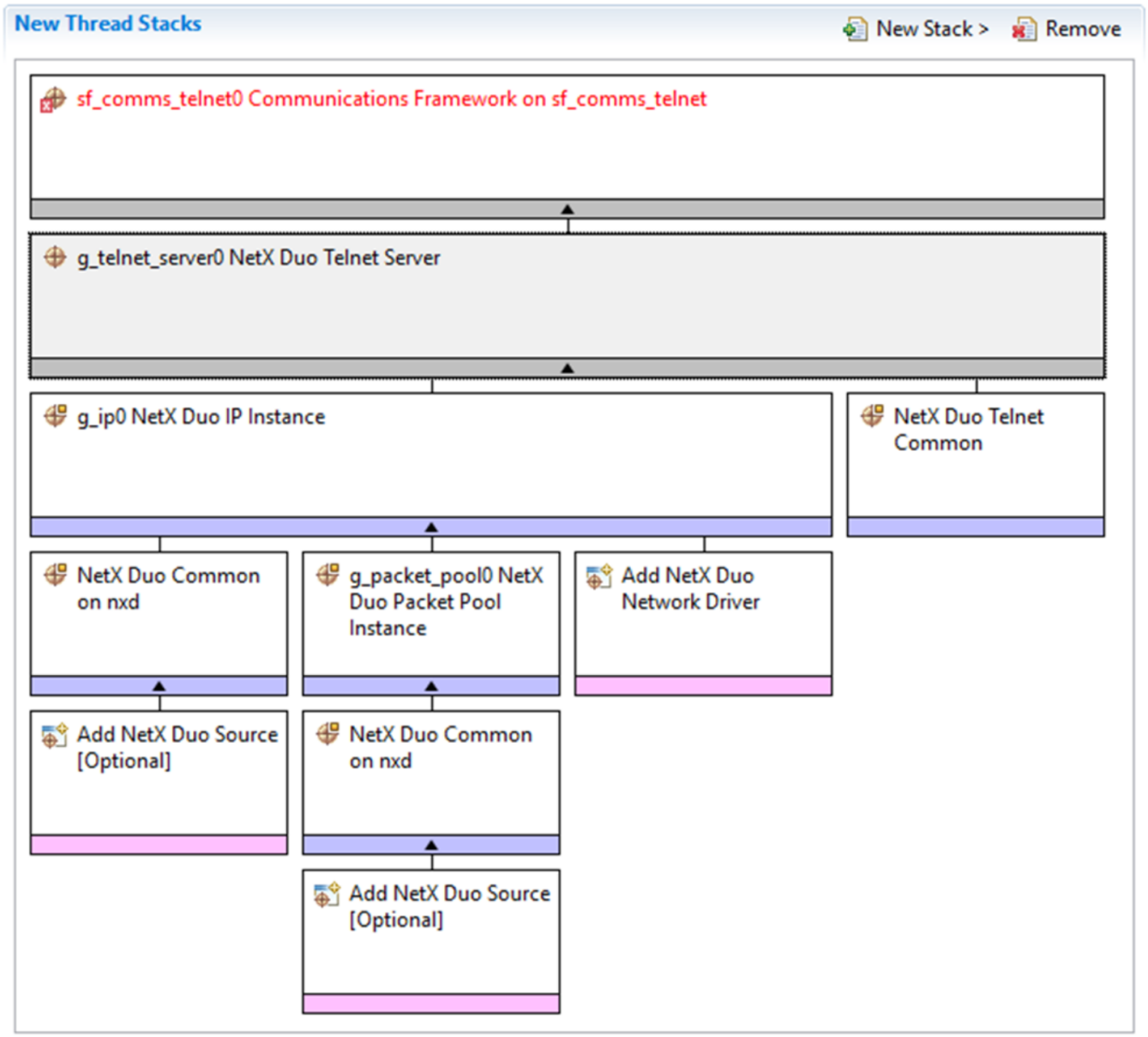


Figure 160: Telnet Communications Framework Module Stack

4.1.13.5 Configuring the Telnet Communications Framework Module

The Telnet Communications Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Telnet Communications Framework Module on sf_comms_telnet

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Packet size in pool memory (bytes)	1536	Packet size in pool memory selection.
Packets to allocate in pool memory (units)	5	Packets to allocate in pool memory selection.
Timeout for internal options (ticks)	10	Timeout for internal options selection.
Maximum number of instances	4	Maximum instances that can be open at any given time.
Name	sf_comms_telnet0	Module name.
Name of generated initialization function	sf_comms_telnet_init0	Name of generated initialization selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.
Telnet client disconnect callback	NULL	User defined callback invoked when the client is disconnected.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Telnet Communications Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the NetX/NetX Duo Telnet Server on telnet_server

ISDE Property	Value	Description
Internal thread priority	16	Internal thread priority selection.
Maximum clients to serve simultaneously	4	Maximum clients to serve simultaneously selection.
Socket window size (bytes)	2048	Socket window size selection.
Server time out (seconds)	10	Duration internal services will suspend for.
Client inactivity timeout (seconds)	600	Client inactivity duration for disconnection.
Timeout check period (seconds)	60	Client activity timeout check interval.
Option negotiation	Enable, Disable Default: Enable	Option negotiation selection.
Use application packet pool	Enable, Disable Default: Disable	Use application packet pool selection.
Packet size in the pool (bytes)	300	Telnet Server only creates this packet pool if 'Option negotiation' is enabled.
Total packet pool size (bytes)	2048	Telnet Server only creates this packet pool if NX_TELNET_SERVER_OPTION_DISABLE.
Name	g_telnet_server0	Module name.
Thread Stack Size (bytes)	2048	Thread stack size selection.
Name of Client Connect Callback Function	NULL	Name of client connect callback function selection.
Name of Receive Data Callback Function	NULL	Name of receive data callback function selection.
Name of Client Disconnect Callback Function	NULL	Name of client disconnect callback function selection.
Name of generated initialization function	telnet_server_init0	Name of generated initialization function selection.
Auto Initialization	Disable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description

Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP cache storage units	Bytes, Entries Default: Bytes	ARP cache storage units selection.
ARP Cache Size (in storage units)	520	ARP Cache Size in Bytes/Entries selection. Note: 1 Entry = 52 Bytes.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.
Link status change callback	NULL	Link status change callback selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings NetX Telnet Common

ISDE Property	Value	Description
Type of Service for TCP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection.
Server TCP port number	23	Server TCP port number selection.
Time to live	128	Time to live selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common on nxd

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	1568	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Port ETHER

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection.
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection.
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection.
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 PHY reset pin selection.
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection.
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection.
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection.
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection.
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Ethernet interrupt priority selection.
Link status monitoring method	PHY Interrupt (Uses LINKSTA Pin), PHY Polling Default: PHY Polling	Link status monitoring method selection.
Name	g_sf_el_nx	Module name.
Channel	0	Channel selection.
Callback	NULL	Callback selection.
Unknown packet receive Callback	NULL	Unknown packet receive callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Telnet Communications Framework Module Clock Configuration

The Telnet Communications Framework module uses the Ethernet peripheral which uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Telnet Communications Framework Module Pin Configuration

To use the Telnet Communications Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the Telnet Communications Framework Module

Resource	ISDE Tab	Pin selection Sequence
SSI	Pins	Select **Peripherals**> **Peripherals**> **Connectivity:ETHERC**> **ETHERC0.RMI or ETHERC1.RMII**

4.1.13.6 Using the Telnet Communications Framework Module in an Application

The typical steps in using the Telnet Communications Framework module in an application are:

1. Initialize the Communications Framework on NX using the sf_comms_api_t::open API
2. Lock the channel for continuous communications using the sf_comms_api_t::lock API if needed
3. Receive data using the sf_comms_api_t::read API
4. Send data using the sf_comms_api_t::write API
5. Unlock the channel from continuous communication using the sf_comms_api_t::unlock command if needed
6. Close the channel using the sf_comms_api_t::close API

These common steps are illustrated in a typical operational flow in the following figure:

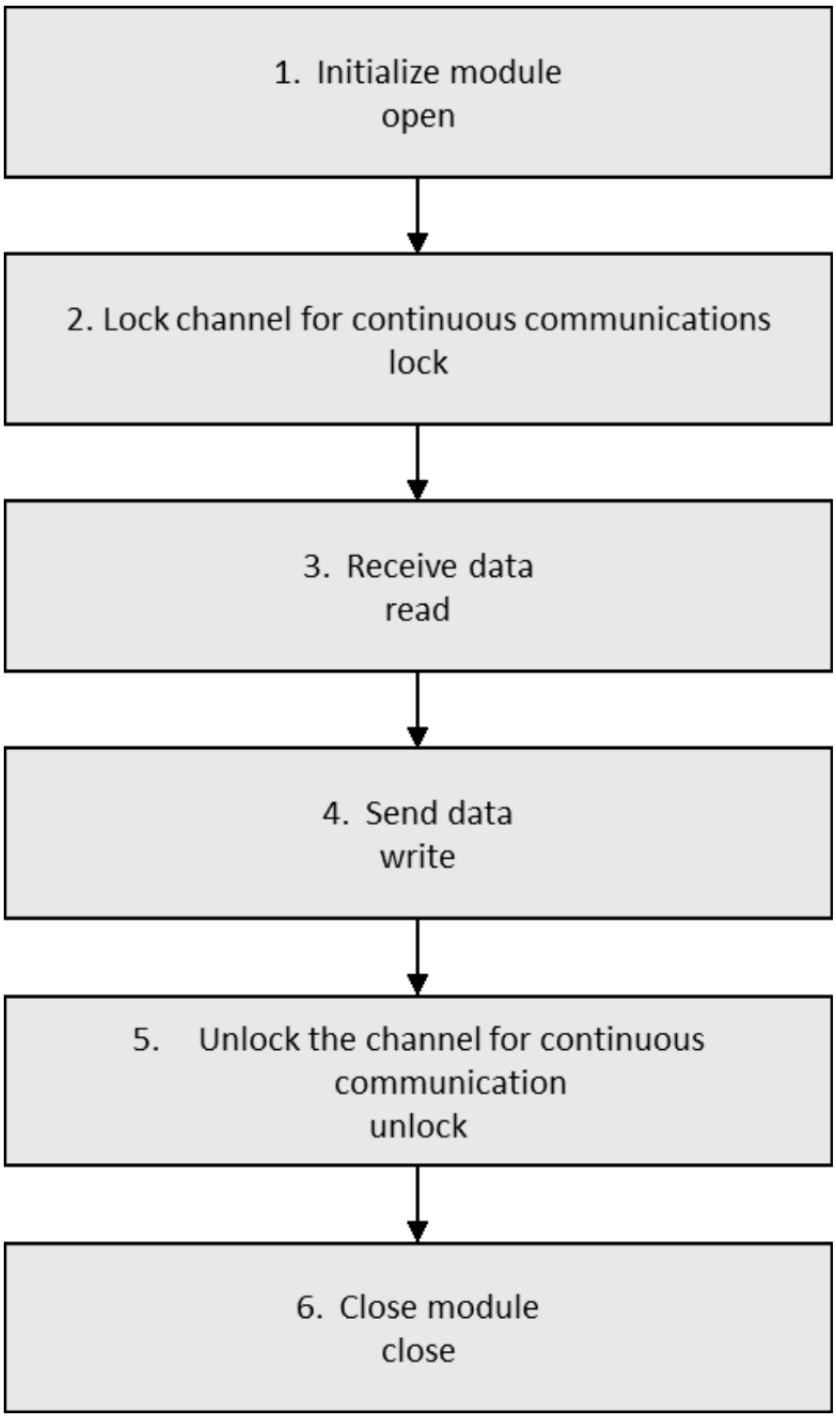


Figure 161: Flow Diagram of a Typical Telnet Communications Framework Module Application

4.1.14 Communications Framework on sf_el_ux_comms_v2

4.1.14.1 Communications Framework on USBX v2 Module Introduction

The Communications Framework on USBX™ (sf_el_ux_comms_v2) implements a high-level API for communications applications that provides an easy-to-use connection over the USB port. The high-level API functions in the framework are compatible with other connection implementations (such as UART and Ethernet), so it is easy to switch from one implementation to another without changing application code. The Communications Framework on USBX uses the USB peripheral on the Synergy MCU device.

Communications Framework on USBX v2 Module Features

- High-level connectivity is supported on USB but is easily changed to UART and Ethernet connectivity without API modification
- Supports channel locking for exclusive access
- Supports USB high speed (HS) or full speed (FS) operation
- Supports data-transfer (DMAC or DTC) peripherals on a Synergy MCU
- ThreadX®-aware implementation uses mutex

Note

Currently, DTC is not supported by both the host and device side driver (only DMAC is supported).

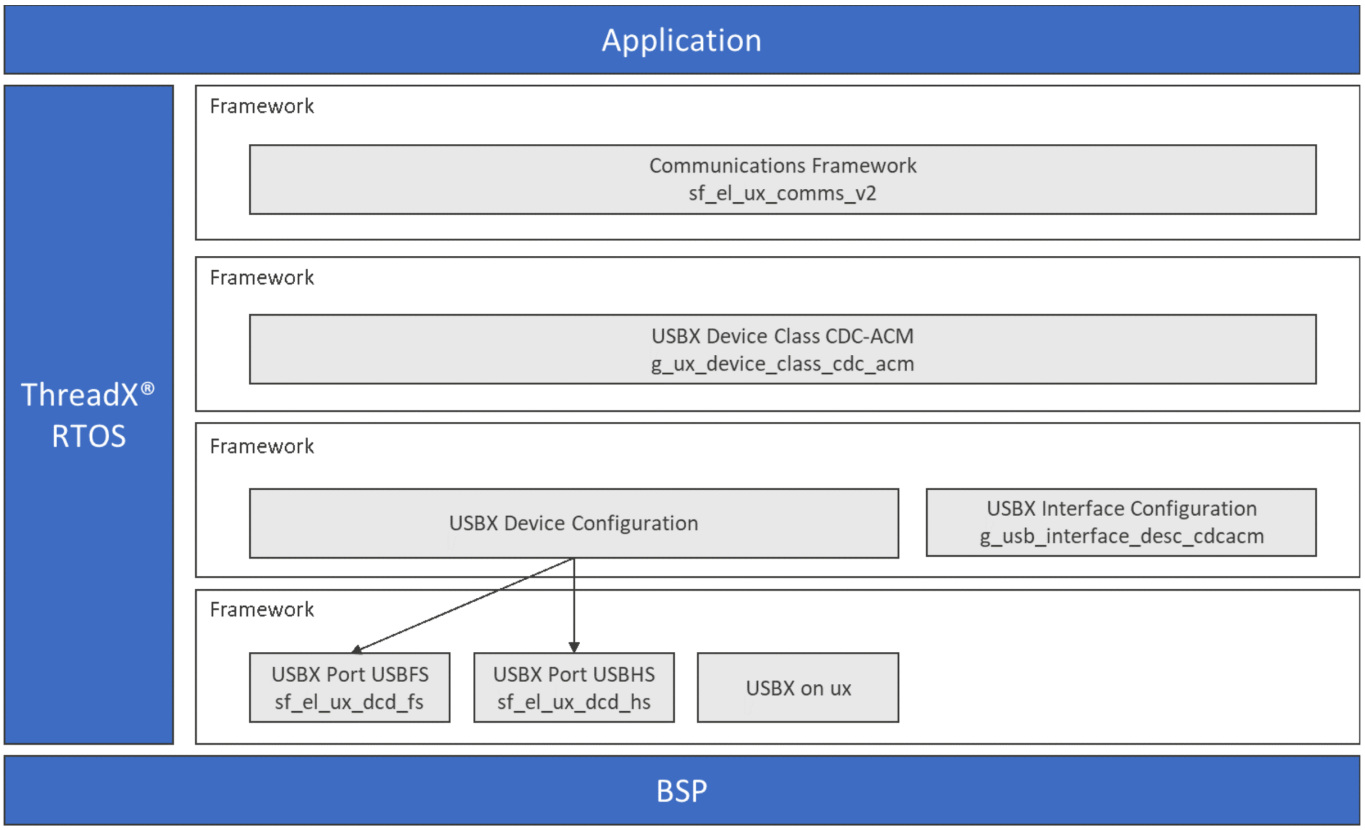


Figure 162: Communications Framework on USBX Module Block Diagram

4.1.14.2 Communications Framework on USBX v2 Module APIs Overview

The Communications Framework on USBX defines API functions for opening, closing, reading and writing over the USB connection. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows

the API summary table.

Communications Framework on USBX Module API Summary

Function Name	Example API Call and Description
open	g_sf_comms0.p_api->open(g_sf_comms0.p_ctrl, g_sf_comms0.p_cfg); Initialize communications driver.
close	g_sf_comms0.p_api->close(g_sf_comms0.p_ctrl); Clean up communications driver.
read	g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl, &destination, bytes, timeout); Read data from communications driver. This call returns after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver or read operation times out.
write	g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl, &source, bytes, timeout); Write data to communications driver. This call returns after all bytes are written or if a timeout occurs while waiting for access to the driver or write operation times out.
lock	g_sf_comms0.p_api->lock(g_sf_comms0.p_ctrl, lock_type, timeout); Lock the communications driver. Reserve exclusive access to the communications driver.
unlock	g_sf_comms0.p_api->unlock(g_sf_comms0.p_ctrl , lock_type); Unlock the communications driver. Release exclusive access to the communications driver.
versionGet	g_sf_comms0.p_api->versionGet(&version); Retrieve the API version in the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Channel opened successfully.
SSP_ERR_IN_USE	Channel already in use.
SSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
SSP_ERR_INTERNAL	Internal error occurs.

SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_NOT_OPEN	Module is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.14.3 Communications Framework on USBX v2 Module Operational Overview

The Communications Framework on USBX provides an easy-to-use connection over the USB port. The high-level API functions in the framework are compatible with other connection implementations (such as UART and Ethernet), so it is easy to switch from one implementation to another without changing applications code. The module uses ThreadX objects like mutex for synchronization for the completion of a transaction. The USBX communication Framework module supports the locking functionality, meaning that the user can lock the communication framework to a thread so that multiple thread can use the same USBX port safely. The locking allows the application to reserve a USB port for a given period of time available between call made to sf_comms_api_t::lock API and sf_comms_api_t::unlock API. The high-level APIs are used to sf_comms_api_t::read API, sf_comms_api_t::write API to support receive or send data to host over USBX CDC-ACM communication interface.

Communications Framework on USBX v2 Module Important Operational Notes and Limitations

Communications Framework on USBX v2 Module Operational Notes

The USBX driver can be implemented on either the HS or FS USB peripherals, depending on the version supported by the target MCU.

A pre-defined weak callback function is available for the USBX CDC-ACM instance activate and another for the USBX CDC-ACM instance deactivate. The user can override these two callback functions if required, by defining a user function using the callback function name given in the Synergy configurator.

Communications Framework on USBX v2 Module Limitations

Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.1.14.4 Including the Communications Framework on USBX v2 Module in an Application

This section describes how to include the Communications Framework on USBX Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Communications Framework on USBX Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

Communications Framework on USBX Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_sf_comms0 Communications Framework on sf_el_ux_comms_v2	Threads	**New Stack> Framework> Connectivity> Communications Framework on sf_el_ux_comms_v2**

When the Communications Framework on USBX Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

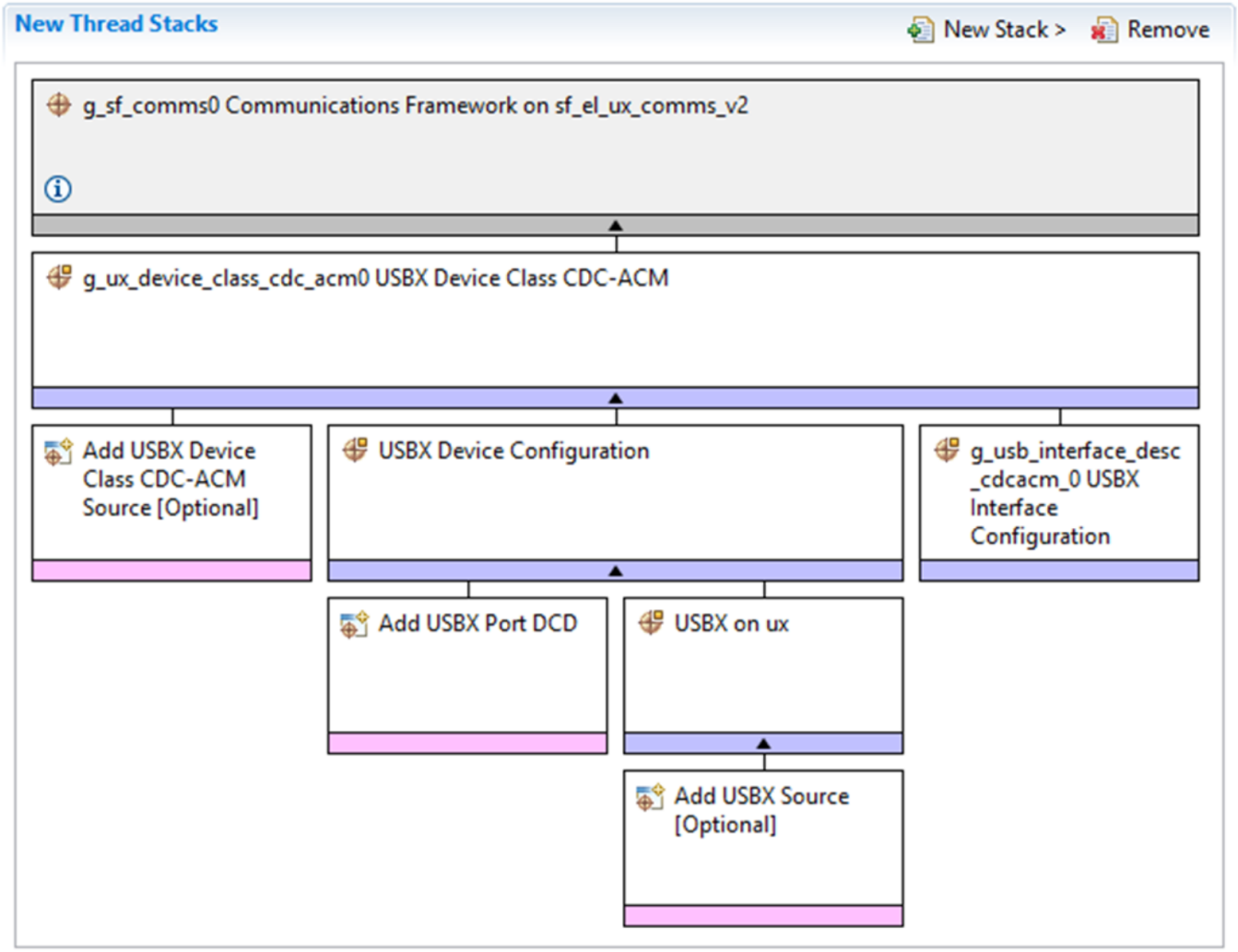


Figure 163: Communications Framework on USBX Module Stack

4.1.14.5 Configuring the Communications Framework v2 on USBX Module

The Communications Framework on USBX Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Communications Framework on USBX Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Read Input Buffer Size (Bytes)	128	Maximum number of bytes that can be received at a time in the read() API.
Timeout in ticks	1000	Timeout value to suspend a USBX CDC instance creation in the open() API.
Name	g_sf_comms0	Module name.
Name of the generated initialization function	sf_comms_init0	Name of helper function to initialize Communications Framework. The function will be presented in the auto-generated code in the <xxx_thread>.c, where <xxx_thread> is the name of your thread symbol given to the Thread property. The function is to be called in the auto-generated code if Auto sf_comms Initialization property is Enabled. If Disabled, the function can be called in the user application.

Auto Initialization	Enable, Disable Default: Enable	Auto Initialization support of Communications Framework. The helper function above will be called in the auto-generated code if this configuration is enabled. Else, the function will not be called automatically and user can call it sometime later.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Communications Framework on USBX v2 Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the USBX Device Class CDC-ACM Module

ISDE Property	Value	Description
Name	g_ux_device_class_cdc_acm0	Specify the name of the USBX Device CDC-ACM Class module instance. It must be a valid C symbol.
USBX CDC-ACM instance_activate Function Callback	ux_cdc_device0_instance_activate	Specify the name of the instance_activate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class" for more information about the instance_activate callback function.

USBX CDC-ACM instance_deactivate Function Callback	ux-cdc_device0_instance_deactivate	Specify the name of the instance_deactivate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class" for more information about the instance_activate callback function.
Name of generated initialization function	ux_device_class_cdc_acm_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Device Configuration Instance

ISDE Property	Value	Description
Vendor ID	0x045B	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (idVendor).
Product ID	0x0000	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (idProduct).
Device Release Number	0x0000	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (bcdDevice).
Index of Manufacturing String Descriptor	0x00	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iManufacturer). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.

Index of Product String Descriptor	0x00	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iProduct). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Index of Serial Number String Descriptor	0x00	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iSerialNumber). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor specific Default: Communications(CDC)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (bDeviceClass).
Index of String Descriptor describing this configuration	0x00	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (iConfiguration). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (wTotalLength).

Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (bNumInterfaces).
Self-Powered	Enable, Disable Default: Enable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit6).
Remote Wakeup	Enable, Disable Default: Disable	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit5).
Maximum Power Consumption (in 2mA units)	50	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2 mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (bMaxPower).
Supported Language Code	0x0409	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.
Name of USBX String Framework	NULL	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.

Total index number of USB String Descriptors in USB String Framework	0	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USBX Language Framework	NULL	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.
Number of Languages to support (US English is applied if zero is set)	0	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.
Name of generated initialization function	ux_device_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Interface Configuration CDC-ACM Instance

ISDE Property	Value	Description
Name	g_usb_interface_desc_cdcacm_0	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.

Interface Number of Communications Class interface	0x00	Specify the index number of Communications Class interface. This configuration is a part of the USB Interface Descriptor (bInterface). The number must not be duplicated with the Interface Number of Data Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Interrupt Transfer endpoint to use for Communications Class	Endpoint 1-9 Default: Endpoint 3	Specify the Endpoint Number of Interrupt Endpoint. It must not be duplicated with ones for the other Endpoints.
Polling period for Interrupt Endpoint (in mS/125us units for FS/HS)	0x0F	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt Endpoint and ignored for Bulk Endpoints. Value is in frame counts (1 ms units for FS mode and 125 us units for HS mode).
Interface Number of Data Class interface	0x01	Specify the index number of Data Class interface. This configuration is a part of the USB Interface Descriptor (bInterface). The number must not be duplicated with the Interface Number of Communications Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Bulk In Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 1	Specify the Endpoint Number of Bulk In Endpoint. It must not be duplicated with ones for the other Endpoints.
Bulk Out Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 2	Specify the Endpoint Number of Bulk Out Endpoint. It must not be duplicated with ones for the other Endpoints.

Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface)	0x00	Specify the index number of String Descriptor Describing Communications Class interface. This configuration is a part of the USB Interface Descriptor (iInterface). Set '0' if do not have String information for the interface.
Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface)	0x00	Specify the index number of String Descriptor Describing Data Class interface. This configuration is a part of the USB Interface Descriptor (iInterface). Set '0' if do not have String information for the interface.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.

Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Communications Framework on USBX v2 Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48 MHz operation. In the SSP configuration window, select the **Clocks** tab to view the clock-source setting.

Communications Framework on USBX v2 Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals are available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence

USBFS	Pins	Select **Peripherals**> **Connectivity: USBFS**> **USBFS0**
USBHS	Pins	Select **Peripherals**> **Connectivity: USBHS**> **USBHS0**

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode.
USBDP	USBDP	USBDP pin.
USBDM	USBDM	USBDM pin.
OVRCURB	None	OVRCURB pin.
OVRCURA	None	OVRCURA pin.
VBUSEN	None	VBUSEN pin.
VBUS	None, P407 Default: P407	VBUS pin.
EXICEN	None	EXICEN pin.
ID	None	ID pin.
VCCUSB	VCCUSB	VCCUSB pin.
VSSUSB	VSSUSB	VSSUSB pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode.
USBHSDP	USBHSDP	USBHSDP pin.
USBHSDM	USBHSDM	USBHSDM pin.

OVRCURB	None	OVRCURB pin.
OVRCURA	None	OVRCURA pin.
VBUSEN	PB00	VBUSEN pin.
VBUS	PB01	VBUS pin.
EXICEN	None	EXICEN pin.
ID	None	ID pin.
USBHSRREF	USBHSRREF	USBHSRREF pin.
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin.
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin.
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin.
VCCUSBHS	VCCUSBHS	VCCUSBHS pin.
VSS1USBHS	VSS1USBHS	VSS1USBHS pin.
VSS2USBHS	VSS2USBHS	VSS2USBHS pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.1.14.6 Using the Communications Framework on USBX v2 Module in an Application

The typical steps in using the Communications Framework on USBX V2 module in an application are:

1. Initialize the Communications Framework on USBX V2 using the sf_comms_api_t::open API.

2. Lock the channel for continuous communications using the sf_comms_api_t::lock API if needed.

3. Receive data using the sf_comms_api_t::read API.

4. Send data using the sf_comms_api_t::write API.

5. Unlock the channel from continuous communication using the sf_comms_api_t::unlock command if needed.

6. Close the channel using the sf_comms_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

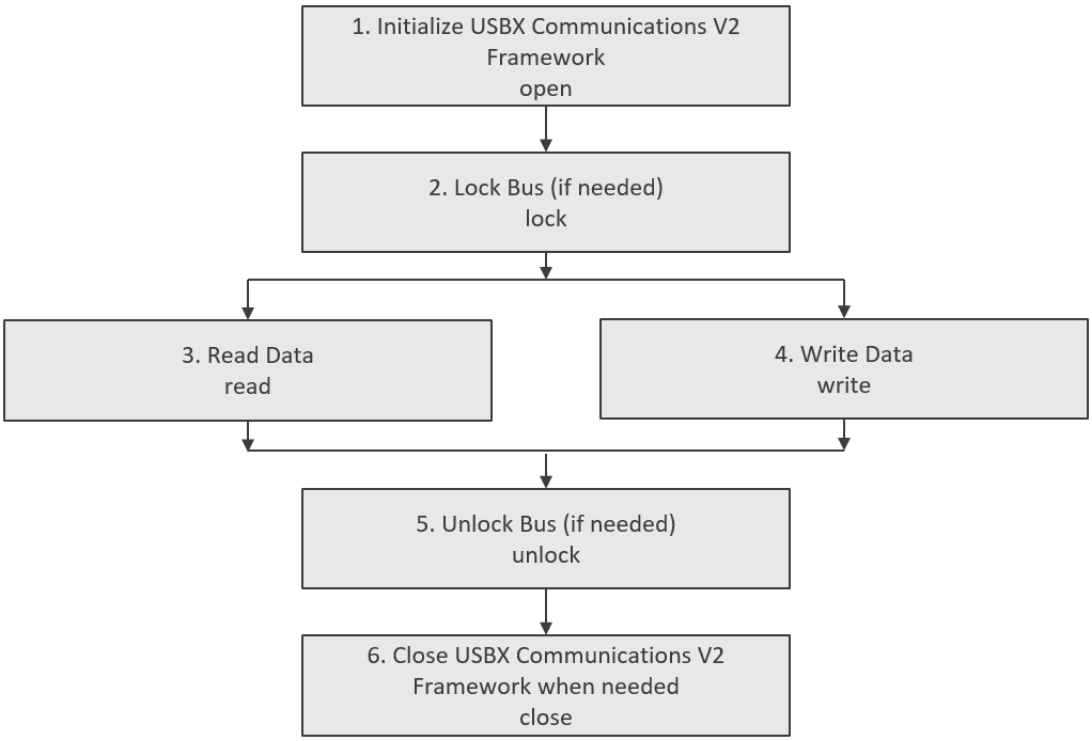


Figure 164: Flow Diagram of a Typical Communications Framework on USBX Module Application

4.1.15 Console Framework

4.1.15.1 Console Framework Introduction

The Console Framework provides a complete API implementation for a menu-driven console command line interface (CLI) using the ThreadX RTOS. The Console Framework module uses a lower-level communications interface, which connects to a hardware option for either UART, USB or Ethernet Telnet connectivity. The Console Framework module has a user-defined menu of commands and various APIs to present a prompt, identify and issue a callback for menu commands and read, write and parse input strings.

Console Framework Module Features

The console framework supports the following features:

- Creation of a menu-based command-line interface
- Submenus and navigation through multiple menus in a single call
- Menu navigation to go up to the parent menu or back to the root
- A help menu for each menu
- Writing NULL terminated strings and reading until return character is received
- An API to help parse arguments to the command line
- Case-insensitive inputs

The Console Framework module organization, as depicted in the thread stack window in the SSP configurator, is shown in the following figure. Each implementation choice, Ethernet, UART, and USB

has its own lower-level modules that are added automatically based on the developer's implementation choice. In most cases, all the needed configuration information is automatically added to the modules leaving the developer with just a few important configuration settings that need to be selected.

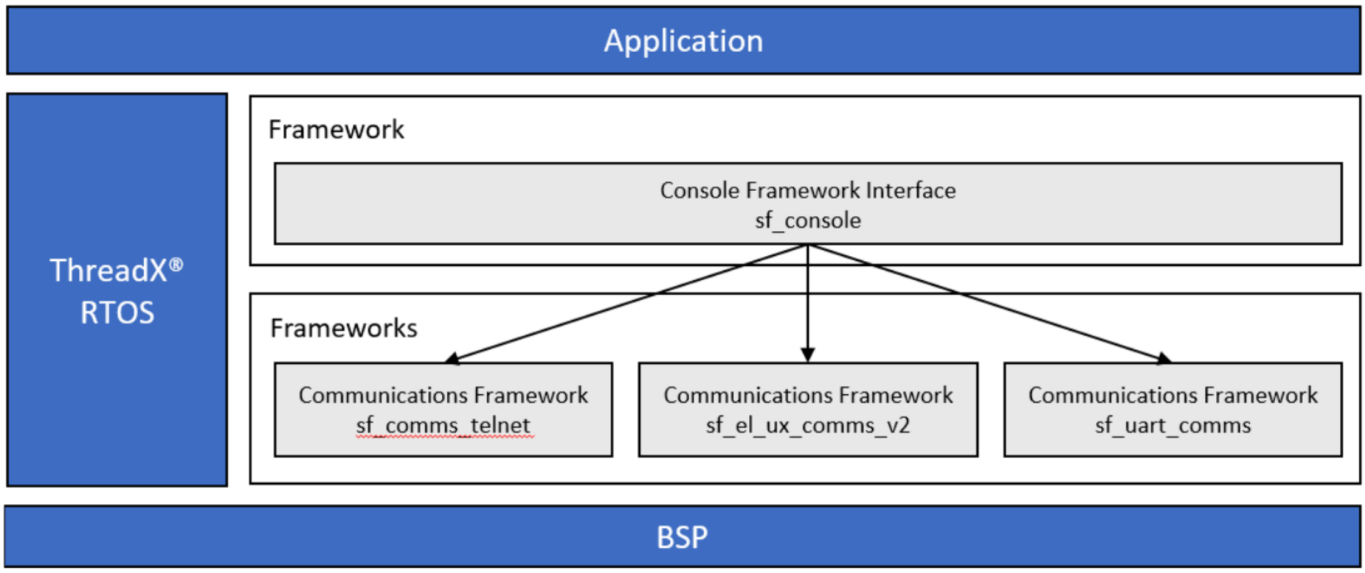


Figure 165: Console Framework Module Block Diagram

4.1.15.2 Console Framework Module APIs Overview

The Console Framework defines APIs for opening, closing, reading, writing and issuing an input prompt. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Console Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_console0.p_api->open(g_sf_console0.p_ctrl, g_sf_console0.p_cfg); The open API configures the console. This function must be called before any other console functions. *Note* *This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the console, open will not need to be called.*
close	g_sf_console0.p_api->close(g_sf_console0.p_ctrl); The close API handles the clean-up of internal driver data.

prompt	g_sf_console0.p_api->prompt(g_sf_console0.p_ctrl, NULL, TX_WAIT_FOREVER); The prompt API prints the prompt string from the menu, waits for input, parses the input based on the menu, and calls the appropriate callback function if a command is identified.
parse	g_sf_console0.p_api->parse(g_sf_console0.p_ctrl, commands, input, s_length); The parse API looks for an input string in the command menu and, if one is found, calls the appropriate callback function.
read	g_sf_console0.p_api->read(g_sf_console0.p_ctrl, ch, 1, TX_WAIT_FOREVER); The read API puts data into the destination, byte-by-byte and echoes the input to the console. Backspace, delete, and left/right arrow keys are supported. Read completes when a line ending with CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes allowed. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, the read will return an error code.
write	g_sf_console0.p_api->write(g_sf_console0.p_ctrl, (uint8_t*)data_string, TX_WAIT_FOREVER); The write API gets the buffer mutex object and handles data transmission at the HAL layer. It obtains the event flag to synchronize the completion of a data transfer.
argumentFind	g_sf_console0.p_api->argumentFind("LED", p_args->p_remaining_string, NULL, &led_num); The argumentFind API locates a command line argument in an input string and returns the index of the character immediately following the argument. Any string numbers are converted to integers.
versionGet	g_sf_console0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	p_ctrlis NULL.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.15.3 Console Framework Module Operational Overview

The Console Framework module is a ThreadX-aware Command Line Interface (CLI). The module uses ThreadX objects like mutex for blocking and synchronization techniques like event flags for the completion of a transaction. The key operational elements of the Console Framework are initialization and input processing, each of which are described in the following sections.

Console Framework Module Initialization

The open call is automatically generated by the ISDE and is in the file where the module was added. The open call requires the application to define a root menu with a variable name that matches the one in the configurator (g_sf_console_root_menu) by default. By the time execution reaches the thread entry function the module is ready to use, provided the necessary hardware connection is established.

Console Framework Module Input Processing

The Console Framework module requires a set of menus, command structures, and callbacks. The Console Framework module typically operates from the prompt, often located within a while loop in the entry thread. The framework sf_console_api_t::prompt API will print the current menu as a prompt, then read input and echo it back to the console (unless echo is disabled in the properties.)

Following operations are performed against the user input:

While the console is accepting input,

- Backspace will remove characters before the cursor.
- Delete will remove characters after the cursor.
- The left and right arrow keys move the cursor.
- The up-arrow key will fill in the last command only when nothing else has been entered.

Note

There is no history beyond the last command; if the up-arrow key is pressed twice, the console does not know what command was entered prior to the last command and it will continue to display the last command.

When the console sees a return character on the read input, it parses the input string and calls the associated callback or switches to the next menu if SF_CONSOLE_CALLBACK_NEXT_FUNCTION is used in place of the callback for the command. The console will continue parsing until a callback function is called. If sf_console_api_t::prompt API is called again, it will prompt using the menu that contains the callback function. To navigate up to the parent menu, enter '^'. To navigate to the root menu from any submenu, enter '~'.

The parser will parse the input command till a white space or end of the string.

For example:

With the list of commands[2] = {"echo", "setbitrate"};

1. >echoIs valid
2. >echo<space>Is valid
3. >echo3Is an unsupported command

4. >setbitrateIs valid
5. >setbitrate 9600Results in the bit rate being set to 9600

Creating Console Framework Module Required Structures – The Menu

The Console Framework requires a menu and it is up to the developer to create the structure that is used by the Console Framework to implement the menu. The console menu structure (depicted in the following figure) includes a pointer to the previous menu, (for creating multi-level menus) a name for the menu, the number of commands in the menu, and a pointer to an array of command structures. As seen in the following figure, each entry in the array of commands includes pointers to the command name string, the help command description string, the associated command callback function, and a context parameter provided to the callback.

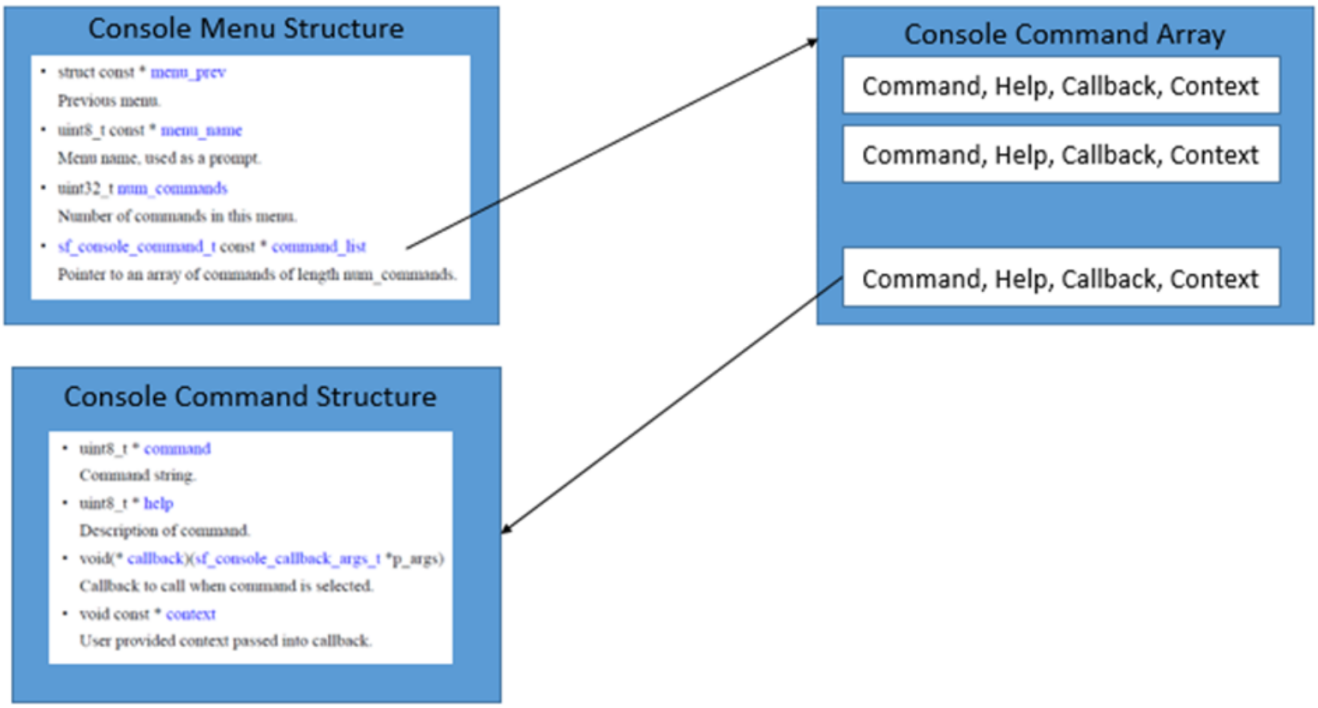


Figure 166: Console Framework Module Menu Structures

The application project example illustrates a Console Framework with a single menu; it controls the toggle of an LED from the CLI. The console command array (g_sf_console_commands, seen on the right in the following figure) stores the array of commands (in this case, just a single command.) The command structure defines the command as "LED TOGGLE", the help description as "Toggle an LED", the callback as led_toggle_callback, and the context as NULL, since it is unused for this example.

The root menu, seen on the left side of the following figure, is identified by the g_sf_console_root_menu structure. The structure defines the menu_prev entry as NULL, since there is only the single menu, the menu_name as "Root", the num_commands as the size of the array divided by the size of an entry (to determine the total number of entries) as "1", and the command_list starting address as "address", the location of the first entry in the command array.

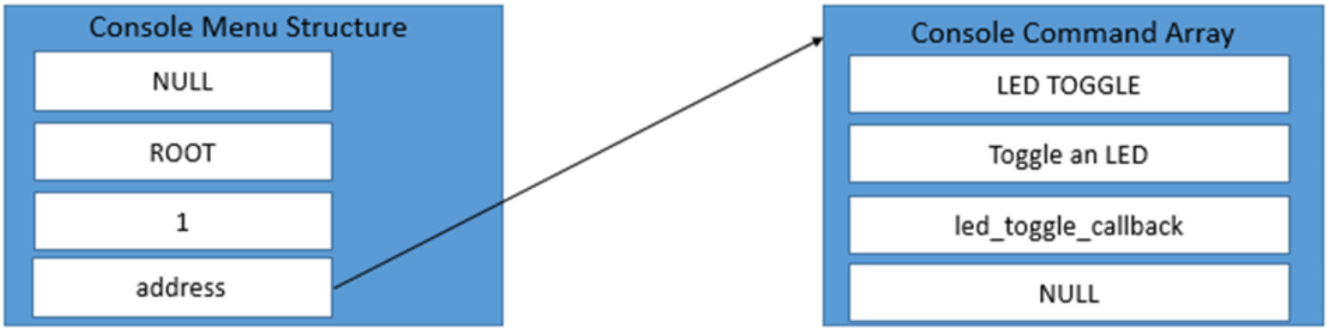


Figure 167: Console Framework Module Menu Structure Example Diagram

Console Framework Module Important Operational Notes and Limitations

Console Framework Module Operational Notes

- To use the Console Framework module sf_console_api_t::prompt API, first set up the menu, command structures and callbacks.

Console Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.15.4 Including the Console Framework Module in an Application

This section describes how to include the Console Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Console Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Console Framework module is g_sf_console0. This name can be changed in the associated Properties window.)

Console Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_console0 Console Framework on sf_console	Threads	**New Stack> Framework> Services> Console Framework on sf_console**

When the Console Framework module on sf_console is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

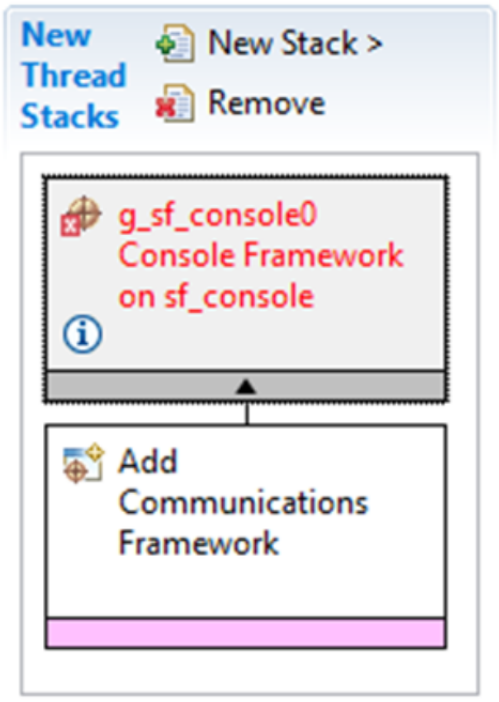


Figure 168: Console Framework Module Stack

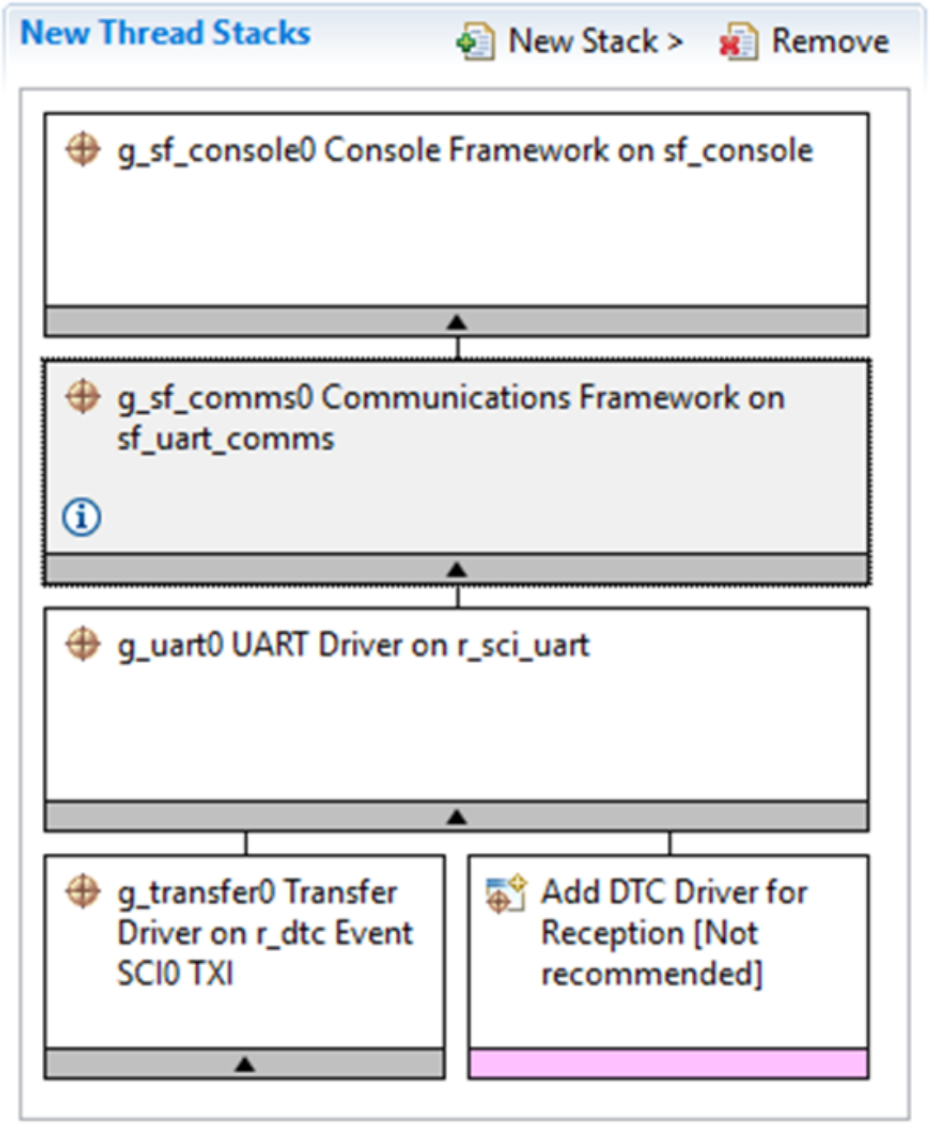


Figure 169: Example Console Framework with UART Communications Framework

4.1.15.5 Configuring the Console Framework Module

The Console Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Console Framework Module on sf_console

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Maximum Input String Length	128	Maximum input string length selection.
Maximum Write String Length	128	Maximum input string length selection.
Console print timeout value	0xFFFFFFFF	Console print timeout value selection.
Name	g_sf_console0	Module name.
Name of Initial Menu (Application Defined)	g_sf_console_root_menu	Initial menu name.
Echo	True, False Default: True	Echo selection.
Autostart	True, False Default: False	Autostart selection.
Name of generated initialization function	sf_console_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto Initialization selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuring the Console Framework Lower-Level Modules

The Console Framework provides three different options for Communication Framework implementations: the Telnet Communications Framework, the Communications Framework on USBX v2, and the UART Communications Framework. Each of these frameworks have several lower-level module options themselves; configuration information for these frameworks is provided in the module overviews for the sf_comms_telnet, sf_el_ux_comms_v2, and sf_uart_comms modules.

4.1.15.6 Using the Console Framework Module in an Application

The typical steps in using the Console Framework module in an application are:

1. Create menu and command structures.
2. Implement needed callbacks.
3. Initialize the Console Framework using hte sf_console_api_t::open API.
4. Use the sf_console_api_t::prompt API to generate the prompt and process commands.
5. Use other APIs (sf_console_api_t::read, sf_console_api_t::write, sf_console_api_t::parse or sf_console_api_t::argumentFind) as needed to process commands.
6. Use sf_console_api_t::close API to close the module if desired.

These common steps are illustrated in a typical operational flow in the following figure:

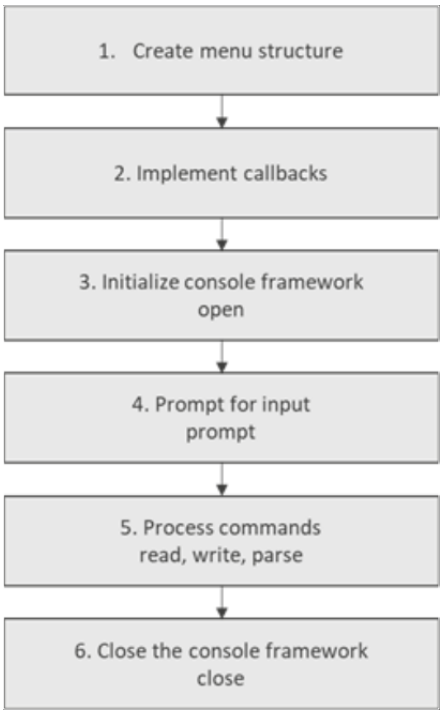


Figure 170: Flow Diagram of a Typical Console Framework Module Application

4.1.16 Crypto Framework

4.1.16.1 Crypto Framework Introduction

The Crypto Framework layer is composed of multiple Crypto modules providing varied cryptographic services. It includes:

SF_CRYPTO for resource synchronization between the crypto modules.

SF_CRYPTO_TRNG for true random number generation.

SF_CRYPTO_HASH for message digest generation. Provides support for MD5, SHA1, SHA 224, SHA 256 algorithms.

SF_CRYPTO_KEY for Key Generation services. Provides support for AES, ECC and RSA keys.

SF_CRYPTO_CIPHER for encryption and decryption services. Provides support for AES and RSA algorithms.

SF_CRYPTO_SIGNATURE for RSA signature generation and verification services.

SF_CRYPTO_KEY_INSTALLATION for key installation services. Provides support for AES, ECC and RSA keys.

Crypto Framework Module Features

The terms "key wrapping" and "key installation" in the context of SSP are defined as follows:

Key Wrapping: The APIs to generate symmetric keys or asymmetric key pairs on the Synergy platform where the private / secret key is a wrapped key (encrypted key).

Key Installation: User generated private /secret keys on a PC (system outside of the Synergy platform) will be installed (no storage) on the Synergy platform and the wrapped private /secret key returned to the user.

Wrapped keys provide the following advantages:

- The wrapped key can only be used on the Synergy platform (MCU) on which it was generated.
- It cannot be moved to another Synergy platform (MCU).
- Original Key cannot be recovered from the wrapped key.

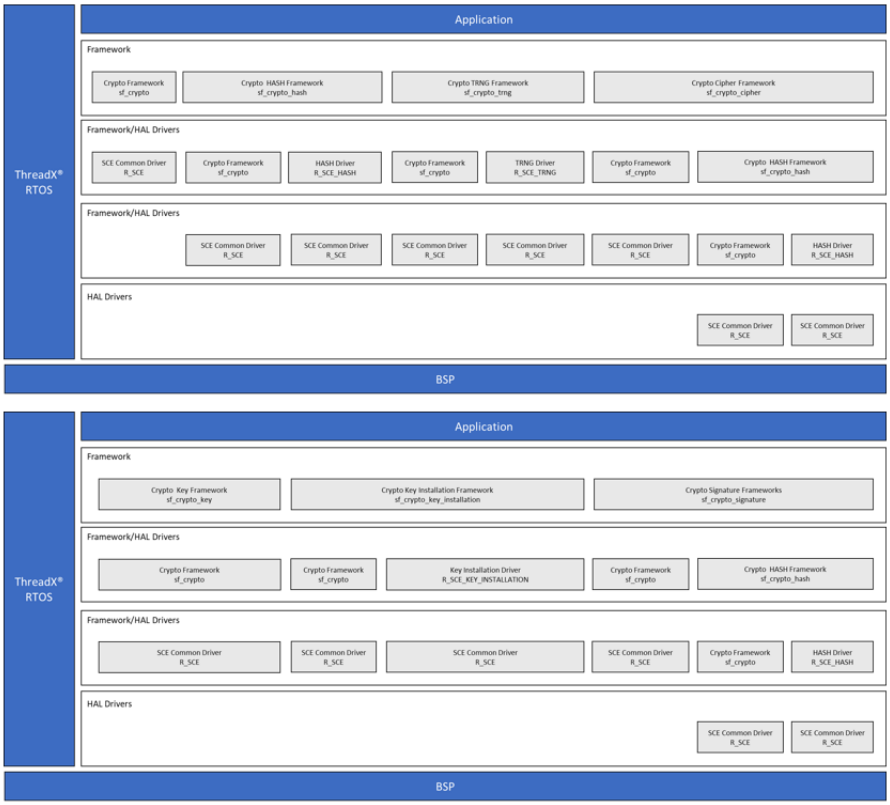


Figure 171: Crypto Framework Module Block Diagram

4.1.16.2 Crypto Framework Module APIs Overview

The Crypto Framework Module defines APIs for a variety of cryptographic services. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Crypto Framework Module API Summary

Function Name	Example API Call and Description

open	g_sf_crypto0.p_api->open(g_sf_crypto0.p_ctrl, g_sf_crypto0.p_cfg); This function is used to open the crypto framework module and the r_sce HAL module.
close	g_sf_crypto0.p_api->close(g_sf_crypto0.p_ctrl); This function is used to close the crypto framework module and the r_sce HAL module.
lock	g_sf_crypto0.p_api->lock(g_sf_crypto0.p_ctrl); This function locks shared resources for cryptography operations.
unlock	g_sf_crypto0.p_api->unlock(g_sf_crypto0.p_ctrl); Unlock shared resources for cryptography operations.
getStatus	g_sf_crypto0.p_api->statusGet(g_sf_crypto0.p_ctrl, &p_status); This function gets the status of the Crypto Framework common module.
versionGet	g_sf_crypto0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto HASH Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_hash0.p_api->open(g_sf_crypto_hash0.p_ctrl, g_sf_crypto_hash0.p_cfg); This function is used to open the crypto hash framework module.
close	g_sf_crypto_hash0.p_api->close(g_sf_crypto_hash0.p_ctrl); This function is used to close the crypto hash framework module.
hashInit	g_sf_crypto_hash0.p_api->hashInit(g_sf_crypto_hash0.p_ctrl); This function initializes the calculation of the hash function. It has to be invoked first and only once, at the beginning of each new calculation.
hashUpdate	g_sf_crypto_hash0.p_api->hashUpdate(g_sf_crypto_hash0.p_ctrl, &p_data_in); This function calculates the hash on the data pointed to by the p_data_in pointer.

hashFinal	g_sf_crypto_hash0.p_api->hashFinal(g_sf_crypto_hash0.p_ctrl, &p_data_in, &p_size); This function finalizes the hash operation on the data pointed to by the p_data_in pointer and stores the output size in the 32-bit word defined by the p_size pointer.
versionGet	g_sf_crypto_hash0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto TRNG Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_trng0.p_api->open(g_sf_crypto_trng0.p_ctrl, g_sf_crypto_trng0.p_cfg); This function is used to open the Crypto TRNG framework module.
close	g_sf_crypto_trng0.p_api->close(g_sf_crypto_trng0.p_ctrl); This function is used to close the Crypto TRNG framework module.
randomNumberGenerate	g_sf_crypto_trng0.p_api->randomNumberGenerate (&p_buffer); This function generates the random number and stores it in memory starting at the address defined by the p_buffer pointer.
versionGet	g_sf_crypto_trng0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto Key Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_key0.p_api->open(g_sf_crypto_key0.p_ctrl, g_sf_crypto_key0.p_cfg); This function is used to open the crypto key framework module.

keyGenerate	g_sf_crypto_key0.p_api->keyGenerate(g_sf_crypto_key0.p_ctrl, &p_secret_key, &p_public_key); This function is used to generate a key or key pair and store it at the p_secret_key and p_public_key pointers.
EcdhSharedSecretCompute	g_sf_crypto_key0.p_api->EcdhSharedSecretCompute(g_sf_crypto_key0.p_ctrl, &p_local_secret_key, &p_remote_public_key, &p_shared_secret); This function is used to perform ECC scalar multiplication
close	g_sf_crypto_key0.p_api->close(g_sf_crypto_key0.p_ctrl); This function is used to close the crypto key framework module and the r_sce HAL module.
versionGet	g_sf_crypto_key0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto Key Installation Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_key_installation.p_api->open(g_sf_crypto_key_installation.p_ctrl, g_sf_crypto_key_installation.p_cfg); This function is used to open the crypto key initialization framework module.
keyInstall	g_sf_crypto_key_installation.p_api->keyInstall(g_sf_crypto_key_installation.p_ctrl, p_user_key_input, p_install_key_input, p_key_data_out); This function is used to install a key using the p_user_key_input, p_install_key_input and p_key_data_out pointers.
close	g_sf_crypto_key_installation.p_api->close(g_sf_crypto_key_installation.p_ctrl); This function is used to close the crypto key installation framework module.
versionGet	g_sf_crypto_key_installation.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto Cipher Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_cipher.p_api->open(g_sf_crypto_cipher.p_ctrl, g_sf_crypto_cipher.p_cfg); This function is used to open the crypto cipher framework module.
cipherInit	g_sf_crypto_cipher.p_api->cipherInit(g_sf_crypto_cipher.p_ctrl, mode, p_key, p_params); This function is used to initialize the crypto cipher framework module.
signUpdate	g_sf_crypto_cipher.p_api->cipherUpdate(g_sf_crypto_cipher.p_ctrl, p_datain, p_dataout); This function performs the cipher encrypt or decrypt operation. It can be called multiple times on additional blocks of data. Result is placed in p_dataout.
cipherFinal	g_sf_crypto_cipher.p_api->cipherFinal(g_sf_crypto_signature.p_ctrl, p_datain, p_dataout); This function performs the final encrypt or decrypt operation. Result is placed in p_dataout.
cipherAadUpdate	g_sf_crypto_cipher.p_api->cipherAadUpdate(g_sf_crypto_cipher.p_ctrl, p_aad); This function updates AAD (Additional Authenticated Data) for AES GCM operation using the p_aad pointer to the AAD data and length. It can be called multiple times on additional blocks of data.
close	g_sf_crypto_cipher.p_api->close(g_sf_crypto_cipher.p_ctrl); This function is used to close the crypto signature framework module.
versionGet	g_sf_crypto_cipher.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Crypto Signature Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_crypto_signature.p_api->open(g_sf_crypto_signature.p_ctrl, g_sf_crypto_signature.p_cfg); This function is used to open the crypto signature framework module.
contextInit	g_sf_crypto_signature.p_api->contextInit(g_sf_crypto_signature.p_ctrl, mode, p_params, p_key); This function is used to initialize the crypto signature framework module.
signUpdate	g_sf_crypto_signature.p_api->signUpdate(g_sf_crypto_signature.p_ctrl, p_message); This function performs the signature update operation. It can be called multiple times to accumulate the message to be signed.
verifyUpdate	g_sf_crypto_signature.p_api->verifyUpdate(g_sf_crypto_signature.p_ctrl, p_message); This function performs the signature verification update operation. It can be called multiple times to accumulate the message whose signature is to be verified.
signFinal	g_sf_crypto_signature.p_api->signFinal(g_sf_crypto_signature.p_ctrl, p_message, p_dest); This function generates the signature and writes it to the destination.
verifyFinal	g_sf_crypto_signature.p_api->verifyFinal(g_sf_crypto_signature.p_ctrl, p_signature, p_message); This function performs the signature verify operation.
close	g_sf_crypto_signature.p_api->close(g_sf_crypto_signature.p_ctrl); This function is used to close the crypto signature framework module.
versionGet	g_sf_crypto_signature.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

The Framework APIs return common SSP error codes and in addition may return error codes from low level modules. Crypto Framework specific error codes can be found in the API Reference section for the specific function.

4.1.16.3 Crypto Framework Module Operational Overview

The following notes apply to SF_CRYPTO, SF_CRYPTO_TRNG, SF_CRYPTO_HASH, SF_CRYPTO_KEY, SF_CRYPTO_KEY_INSTALLATION, SF_CRYPTO_CIPHER and SF_CRYPTO_SIGNATURE modules.

Endianness configuration parameter

- In the Synergy Configurator (if *only* the Crypto HAL drivers are selected without the Crypto Framework modules), endianness is a configurable parameter and set to *big endian* by default. This is the mode that was supported in previous releases.
- In the Synergy Configurator, if Crypto Framework modules are selected (the respective HAL component is included automatically), the endianness flag is locked to the *little endian* mode where only the byte array format is supported.
- The Crypto Framework Layer APIs support only byte arrays for input and output data.
- The Crypto HAL APIs support WORD / 32-bit (uint32_t) arrays for input and output data.
- Users with data in byte array format should use the Crypto Framework modules.
- For any existing projects using the Crypto HAL API without the respective Framework API, Big Endian data may have been used and in that case, it needs to be converted to little Endian to use framework API or continue to use the existing HAL API directly.
- If using the HAL APIs directly, the user needs to make sure that the data cast as (uint32_t) matches the endianness of the HAL module used.

Alignment for data buffers

- All data buffers allocated to be passed in as input buffers must be aligned on WORD boundary.

Blocking calls

- All Crypto Framework APIs are blocking calls.

SF CRYPTO Framework Services

- The SF CRYPTO Framework modules include:
    - SF_CRYPTO for HW initialization and resource synchronization between the crypto modules.
    - SF_CRYPTO_TRNG for true random number generation.
    - SF_CRYPTO_HASH for message digest generation. Provides the ability to process data in chunks before finalizing the hash operation.
    - SF_CRYPTO_KEY for RSA, AES, ECC. Provides the option to generate wrapped keys in addition to plain text keys.
    - SF_CRYPTO_KEY_INSTALLATION provides key installation services for RSA, AES and ECC keys.
    - SF_CRYPTO_CIPHER provides encryption and decryption services for AES and RSA keys.
    - SF_CRYPTO_SIGNATURE provides RSA signature generation and verification services.

VersionGet API

This API can be called at any time, that is, even before a module is opened.

SF CRYPTO Framework Module Overview Description

The SF CRYPTO Framework provides a high-level API and is implemented on sf_crypto. The SF

CRYPTO Framework provides a foundation for the Framework Crypto services through the Secure Cryptographic Engine (SCE) HAL module.

SF CRYPTO Framework Module Features

- The SF CRYPTO Framework opens/ initializes the underlying HW Secure Crypto Engine.
- Provides services for access to shared resources for other Crypto Framework modules SF_CRYPTO_TRNG, SF_CRYPTO_HASH, SF_CRYPTO_KEY, SF_CRYPTO_CIPHER, SF_CRYPTO_SIGNATURE and SF_CRYPTO_KEY_INSTALLATION.

SF CRYPTO Framework Module Operational Notes

- sf_crypto_api_t::open API is called during initialization because the "Auto Initialization" configuration option is enabled by default.
- sf_crypto_api_t::lock API and sf_crypto_api_t::unlock API are for the SF_CRYPTO_XXX modules to to be used.

However, if the application is making a direct call to HAL APIs, use sf_crypto_api_t::lock API just before making the call to HAL module.

- Use sf_crypto_api_t::unlock API just after returning from the call to HAL module.
- sf_crypto_api_t::statusGet API requires the control block / p_ctrl as input parameter. Hence it should be called only after the open API of SF_CRYPTO module is called.

Configuration Settings for the SF CRYPTO Framework Module

Configuration parameter	Description
uint32_t wait_option	Wait option for RTOS service calls Defines how the service behaves if there is not enough memory available. The wait options are defined as follows: TX_NO_WAIT (0x00000000) T X_WAIT_FOREVER (0xFFFFFFFF) timeout value (0x00000001 through 0xFFFFFFFE) Selecting TX_NO_WAIT results in an immediate return from this service regardless of whether or not it was successful. This is the only valid option if the service is called from initialization. Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until enough memory is available. Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the memory. Default set through ISDE isTX_WAIT_FOREVER.
crypto_instance_t * p_lower_lvl_crypto	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings. Configured by Synergy Configurator.

void const * p_context	Placeholder for user data. For future expansion.
void * p_memory_pool	Byte pool address. Configured by Synergy Configurator.
uint32_t memory_pool_size	Byte pool size. Default set by ISDE is 128 bytes. Caller to allocate pool based on the number of SF_CRYPTO_XXX module instances created. Recommended sizes: In addition to the default byte pool: *Note* *An additional 12 bytes per instance is needed for RTOS housekeeping.* *24 + 12 bytes per instance of SF_CRYPTO_KEY module if RSA Key is required.* *264 +12bytes per instance of SF_CRYPTO_KEY module if AES Key is required.* *SF_CRYPTO_KEY module if AES Key is required.* *112 +12 bytes per instance of SF_CRYPTO_HASH module.* *1200 bytes per instance of SF_CRYPTO_CIPHER module for RSA algorithm is required.* *600 bytes per instance of SF_CRYPTO_CIPHER module for AES algorithm is required.* *1450 bytes per instance of SF_CRYPTO_SIGNATURE module.*
sf_crypto_close_option_t close_option	Close option Selects how the SCE module can be closed. SF_CRYPTO_CLOSE_OPTION_DEFAULT - Close the module only if none of the other SF_CRYPTO_XXX modules are opened. *Note* *This is the default setting.* *SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE - Close the module regardless of SF_CRYPTO_XXX modules status.* *With either option, It is the responsibility of the caller to ensure that the SF CRYPTO module is not closed when any of the other Crypto Framework modules are open.*

SF CRYPTO Framework Module Limitations

It is the responsibility of the caller to ensure that the SF CRYPTO module is not closed when any of the other Crypto Framework modules are open.

Using the SF CRYPTO Framework Module in an Application

The typical steps in using the SF CRYPTO Framework module in an application are:

To use the SF CRYPTO module.

- Ensure that the configuration parameters (given in the previous table) are set as per the needs of the application.
- Use the sf_crypto_api_t::open API to Initialize the SF_CRYPTO and the SCE HAL module (R_SCE) through the SCE common driver. This is done by Synergy Configurator when the Auto Initialize setting is at default.
- The little endian mode is set by default. It is locked and not configurable.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_api_t::close API to close the Crypto Framework services and the HW SCE.

SF CRYPTO TRNG Framework Module Overview Description

The SF CRYPTO TRNG Framework provides a high-level API and is implemented on sf_crypto_trng. The SF CRYPTO TRNG Framework provides True Random Number Generation services through the Secure Cryptographic Engine (SCE) HAL module.

SF CRYPTO TRNG Framework Module Features

- The SF CRYPTO TRNG Framework module uses the TRNG HAL interfaces to the underlying Secure Crypto Engine.
- Access to shared resources through SF CRYPTO Framework module.

Configuration Settings for the SF TRNG CRYPTO Framework Module

Configuration Parameter	Description
sf_crypto_instance_t * p_lower_lvl_common;	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
trng_instance_t * p_lower_lvl_instance;	Pointer to a TRNG HAL driver instance. Configured by Synergy Configurator.
void * p_extend;	Extension parameter for hardware specific settings. For future use.

The configuration is set through the ISDE for the TRNG HAL driver:

Configuration Parameter for the TRNG HAL Module on r_sce_trng

Configuration parameter	Description
uint32_t num_attempts	Number of attempts within which a true random number is to be generated. Set to 2 by default.

SF CRYPTO TRNG Framework Module Limitations

None.

SF CRYPTO HASH Framework Module Overview Description

The SF CRYPTO HASH Framework provides a high-level API and is implemented on sf_crypto_hash. The SF CRYPTO HASH Framework provides hash/message digest services through the Secure Cryptographic Engine (SCE) HAL module.

The hash functions supported are MD5, SHA-1, SHA-224 and SHA-256.

From FIPS Secure Hash Standard:

All of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a *message digest*. These algorithms enable the determination of a message's integrity: any change to the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

For the supported hash functions, the message size should be $< 2^{64}$ bits.

The message digest sizes are as follows:

MD5 : 16 bytes

SHA-1 : 20 bytes

SHA-224: 28 bytes

SHA-256: 32 bytes

SF CRYPTO HASH Framework Module Features

- The SF CRYPTO HASH Framework utilizes the underlying Secure Crypto Engine to provide HASH services.
- This module provides enhancements over the HAL Driver such as:
  - Initializing the HASH value.
  - Processing chunks of data through sf_crypto_hash_api_t::hashUpdate API before finalizing the hash operation.
  - Formatting the final block for the final HASH operation.

Configuration Settings for the SF CRYPTO HASH Framework Module

Configuration Parameter	Description
sf_crypto_hash_type_t hash_type;	HASH algorithm type. Select MD5, SHA1, SHA 224 or SHA256. See the header file for the definitions.
crypto_instance_t * p_lower_lvl_crypto	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
hash_instance_t * p_lower_lvl_instance;	pointer to HASH lower-level module instance. Configured by ISDE.
void * p_extend	Extension parameter for hardware specific settings. Optional. Configured by Synergy Configurator.

SF CRYPTO HASH Framework Module Limitations

None

SF CRYPTO KEY Framework Module Overview Description

The SF CRYPTO KEY Framework provides a high-level API and is implemented on sf_crypto_key. The SF CRYPTO KEY Framework provides cryptographic key generation services through the Secure Cryptographic Engine (SCE) HAL module.

The wrapped keys are often referred to by different names, for example encrypted key, key handle, wrapped key. Key wrapping on Synergy platform is considered secure as those keys cannot be used outside of the platform.

SF CRYPTO KEY Framework Module Features

The following key types can be generated using the services of the SF CRYPTO KEY module:

- RSA 2048-bit, 1024-bit plain text / raw keys.
- RSA 2048-bit, 1024-bit standard format wrapped private keys (public keys in plain).
- AES 128-bit, 192-bit and 256-bit wrapped keys for ECB, CBC, CTR and GCM chaining modes.
- AES 128-bit and 256-bit wrapped keys for XTS chaining mode.
- ECC 192-bit, 224-bit, 256-bit, 384-bit plain-text and wrapped keys.

SF CRYPTO KEY Framework Module Operational Notes

AES Keys

AES wrapped key sizes are as follows:

```
/* Return Wrapped AES secret key size in bytes for a 128-bit AES Key */
#define AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES  (36U)
/* Return Wrapped AES secret key size in bytes for a 192-bit AES Key */
#define AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return Wrapped AES secret key size in bytes for a 256-bit AES Key */
#define AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return Wrapped AES-XTS secret key size in bytes for a 128-bit AES XTS Mode Key */
#define AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return AES-XTS secret key size in bytes for a 256-bit AES XTS Mode Key */
#define AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES  (84U)
```

RSA Keys

The format of RSA plain text keys generated by the keyGenerate API is as follows:

RSA Public key

Byte 0 to Byte 3: Public key exponent

Byte 4 : Start of RSA modulus

(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private key in standard format

Byte 0: Private key exponent (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

Followed by RSA modulus. (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private key in CRT format

The components are ordered in the following order with exponent2 at byte 0:

exponent2 // the second factor's CRT exponent, a positive integer

prime2 // the second factor, a positive integer

exponent1 // the first factor's CRT exponent, a positive integer

prime1 // the first factor, a positive integer

coefficient // the (first) CRT coefficient, a positive integer

The format of RSA wrapped keys generated by the keyGenerate API is as follows:

RSA Public key is always in plain text.

Byte 0 to Byte 3: Public key exponent

Byte 4 : Start of RSA modulus

(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private key in standard format

Byte 0: Private key exponent is wrapped. (Length is 148 bytes for RSA 1024-bit and 276 bytes for RSA 2048-bit keys)

Followed by RSA modulus in plain text. (Length is 128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

Configuration Settings for the SF CRYPTO KEY Framework Module

Configuration parameter	Description
sf_crypto_key_type_t key_type	Key type to be generated. Plain text or Wrapped keys. See the header file for the definitions.
sf_crypto_key_size_t key_size	Key size to be generated. See the header file for the definitions.

sf_crypto_data_handle_t domain_params;	Pointer to domain parameters for the requested key type. Structure contains, pointer to the data (contains the domain data) and data length. Structure contains the domain data in the order a||b||p||n for ECC as defined in FIPS186-3 and data length. To be filled appropriately for ECC and set to NULL for RSA and AES Key types, since this parameter only applies to the ECC function.
sf_crypto_data_handle_t generator_point	Pointer to the generator base point of curve in the order Gx||Gy for ECC (where Gx and Gy are x and y coordinates respectively) and data length. To be set to NULL for RSA and AES key types.
sf_crypto_instance_t * p_lower_lvl_crypto_common	Pointer to a Crypto Framework common instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings (Reserved for future use). Configured by Synergy Configurator.

SF Crypto Signature Framework Module Overview Description

The SF CRYPTO Signature Framework provides a high-level API and is implemented on sf_crypto_signature. This module is in the SSP framework layer that interfaces with Synergy HAL drivers for the hardware level cryptography operations. The module functions specified herein are used to sign / verify messagethat is provided as input.

SF Crypto Signature Module Features

- This module currently supports signature generation and signature-verification for RSA algorithm. There is support for both 1024-bits and 2048-bits key lengths for standard format plain-text, standard format wrapped private key and CRT plain-text keys.
- RSASSA-PKCS1 v1.5 is the supported signature scheme. The input message can be passed as raw message to be signed/verified or can be PKCS1 v1.5 encoded and padded before sign/verify.
- SHA1, SHA224, SHA256 are the supported hashing algorithms for PKCS1 v1.5 encoding /padding schemes.
- This module allows signature generation / verification on data which is smaller than a block size.
- If all of the data is not available at once, update APIs can be used to accumulate the incoming chunks of data and finally sign/verify the message only when all the message is gathered.
- Module allows to switching between sign and verify operations with less expensive API calls which are not involved in allocating and deallocating memory.

SF Crypto Signature Framework Module Operational Notes

1. Operation modes for Signature Framework Module:
    a. SF_CRYPTO_SIGNATURE_MODE_SIGN
    b. SF_CRYPTO_SIGNATURE_MODE_VERIFY
2. Input Message Format to the Signature Framework APIs to perform sign or verify operation:
    a. SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_NONE

b. SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA1_PKCS1_1_5
c. SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA224_PKCS1_1_5
d. SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA256_PKCS1_1_5

Configuration Settings for SF CRYPTO Signature Framework Module

Configuration parameter	Description
sf_crypto_key_type_t key_type;	Type of Key RSA plain text or wrapped. See the header file for the definitions.
sf_crypto_key_size_t key_size	Size of Key RSA 1024-bit /2048-bit key. See the header file for the definitions.
sf_crypto_hash_instance_t * p_lower_lvl_sf_crypto_hash	Pointer to the Crypto Hash framework module instance structure. Configured by Synergy Configurator.
sf_crypto_instance_t * p_lower_lvl_crypto_common	Pointer to Crypto Common module instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings. Optional. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings (Reserved for future use). Configured by Synergy Configurator.

SF CRYPTO Signature Framework Module Limitations

None

SF Crypto Cipher Framework Module Overview Description

The SF CRYPTO Cipher Framework provides a high-level API and is implemented on sf_crypto_cipher. This module is in the SSP framework layer and provides encryption and decryption services through the Secure Cryptographic Engine (SCE) HAL module.

SF Crypto Cipher Module Features

- This module currently supports encryption and decryption for AES and RSA algorithms.
- This module allows encryption/ decryption of data when available in chunks through the sf_crypto_cipher_api_t::cipherUpdate API and final() when the last chunk /all the data is gathered.
- Once the module is opened for a specific key type and key size, the encrypt and decrypt modes can be switched by calling the sf_crypto_cipher_api_t::cipherInit API.

AES algorithm support:

- AES 128-bit, 192-bit and 256-bit plain text and wrapped keys for the following chaining modes:
    - ECB (Electronic Code Book)
    - CBC (Cipher Block Chaining)
    - CTR (Counter Mode)

    - GCM (Galois Counter Mode)
- AES 128-bit and 256-bit plain text and wrapped keys for XTS (XEX-based tweaked code-book mode with ciphertext stealing) are supported.
- IV provided for AES GCM operations can be either 96-bits or a 96-bit IV formatted to 128-bits.
    - Example:
        - 96-bit IV: 94c1935afc061cbf254b936f
        - 96-bit IV formatted to 128-bits: 94c1935afc061cbf254b936f00000001
- PKCS#7 padding scheme is supported for ECB and CBC modes. This scheme can be used when the data is of any block size from 1 to 255 bytes.
- No padding option is supported for all modes where the data is exactly a multiple of the AES block size.
    - For GCM, no padding option is to be selected even when the data is not a multiple of the block size.

RSA algorithm support:

- RSA 1024-bit and 2048-bit plain-text standard format, plain-text CRT format and standard format wrapped keys are supported.
- RSA encrypt operation requires RSA public key and RSA decrypt operation requires RSA private key.
- RSAES-PKCS1-v1_5 (RSA Encryption Scheme)
    - RSA-PKCS1 v1.5 encoding /padding scheme is supported.
    - The input raw message is encoded and formatted to be encrypted. It requires that the message be less than k-11 where k is the length of the modulus of the selected key.
- No padding option should be selected when an encoded and formatted block (exactly the size as the modulus of the selected key) needs to be encrypted / decrypted.

SF Crypto Cipher Framework Module Operational Notes

Configuration Settings for the SF Crypto Cipher Framework Module

Configuration Parameter	Description
sf_crypto_key_type_t key_type	Key type for cipher operation: AES or RSA plain text or wrapped. Please refer to the header file for the definitions.
sf_crypto_key_size_t key_size	Key size for cipher operation: AES 128-bit / 192-bit/256-bit key or RSA 1024-bit /2048-bit key. Please refer to the header file for the definitions.
sf_crypto_cipher_mode_t cipher_chaining_mode	Applicable only to AES algorithm: ECB, CBC, CTR, XTS or GCM chaining modes Set to ECB for RSA. Please refer to the header file for the definitions.
sf_crypto_instance_t * p_lower_lvl_crypto_common	Pointer to Crypto Framework Common module instance. Configured by Synergy Configurator.
sf_crypto_trng_instance_t * p_lower_lvl_crypto_trng	Pointer to Crypto Framework TRNG module instance. Configured by Synergy Configurator.

void	const	* p_extend	Extension parameter for hardware specific settings.Optional. Configured by Synergy Configurator.

SF CRYPTO Cipher Framework Module Limitations

- AES XTS mode only supports lengths which are a multiple of 32-bits / 4 bytes.

SF Crypto Key Installation Framework Module Overview Description

The SF Crypto Key Installation Framework provides a high-level API and is implemented on sf_crypto_key_installation. The SF Crypto Key Installation Framework Key Installation services through the Secure Cryptographic Engine (SCE) HAL module.

Note

1. The API returns the wrapped key (please refer to the SF CRYPTO KEY Module section for the wrapped key format and size details) to the user and does not store the key for future use. It is the responsibility of the application to store the wrapped key in non-volatile memory for future use.
2. This Framework module provides the same level of service as the Crypto HAL module without any enhancements. It is provided for thread safe operation and for completeness of Crypto support in the Framework layer.

SF Crypto Key Installation Framework Module Features

- The SF Crypto Key Installation Framework module uses the KEY INSTALLATION HAL interfaces to the underlying Secure Crypto Engine.
- Access to shared resources through SF CRYPTO Framework module.
- This module supports key installation for the following keys:
  - RSA:
    - 1024-bit and 2048-bit plain text standard format keys.
    - *Note*
      CRT keys are not supported.
  - AES:
    - 128-bit, 192-bit and 256-bit plain text keys for ECB, CBC, CTR and GCM chaining modes.
    - 128-bit and 256-bit plain text keys for XTS chaining mode.
  - ECC:
    - 192-bit and 256-bit plain text keys.
- The session key, iv and shared index must be provided to the key installation API in the specified format.
- Upon installation the key installation service returns a wrapped key to the caller for any future usage of installed key on that device.
- ECC 224-bit and 384-bit keys are supported by the HAL module and not by the framework.

Configuration Settings for the SF Crypto Key Installation Framework Module

Configuration parameter	Description

sf_crypto_key_type_t key_type;	Type of key to be installed. The prepared key will be of the encrypted type. Please refer to the header file for the definitions.
sf_crypto_key_size_t key_size;	Size of key to be installed. Please refer to the header file for the definitions.
sf_crypto_instance_t * p_lower_lvl_common;	Pointer to a Crypto Framework common instance. Configured by Synergy Configurator.
key_installation_instance_t * p_lower_lvl_instance;	Pointer to Crypto Key Install HAL instance Configured by Synergy Configurator.
void const * p_extend;	Extension parameter for hardware specific settings. For future use.

4.1.16.4 Including the Crypto Framework Module in an Application

This section describes how to include the Crypto Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Crypto Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Crypto Framework module is g_sf_crypto0. This name can be changed in the associated Properties window.)

Crypto Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_crypt0 Crypto Framework on sf_crypto	Threads	**New Stack> Framework> Crypto> Crypto Framework on sf_crypto**
g_sf_crypt_key0 Crypto Key Framework on sf_crypto_key	Threads	**New Stack> Framework> Crypto> Crypto Key Framework on sf_crypto_key**
g_sf_crypt0 Crypto TRNG Framework on sf_crypto_trng	Threads	**New Stack> Framework> Crypto> Crypto TRNG Framework**
g_sf_crypt0 Crypto HASH Framework on sf_crypto_hash	Threads	**New Stack> Framework> Crypto> Crypto HASH Framework**
g_sf_crypt_key0 Crypto Key Installation Framework on sf_crypto_key_installation	Threads	**New Stack> Framework> Crypto> Crypto Key Installation Framework on sf_crypto_key_installation**

g_sf_crypt0 Crypto cipher Framework on sf_crypto_cipher	Threads	**New Stack> Framework> Crypto> Crypto cipher Framework**
g_sf_crypt0 Crypto signature Framework on sf_crypto_signature	Threads	**New Stack> Framework> Crypto> Crypto signature Framework**

When the Crypto Framework module on sf_crypto is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

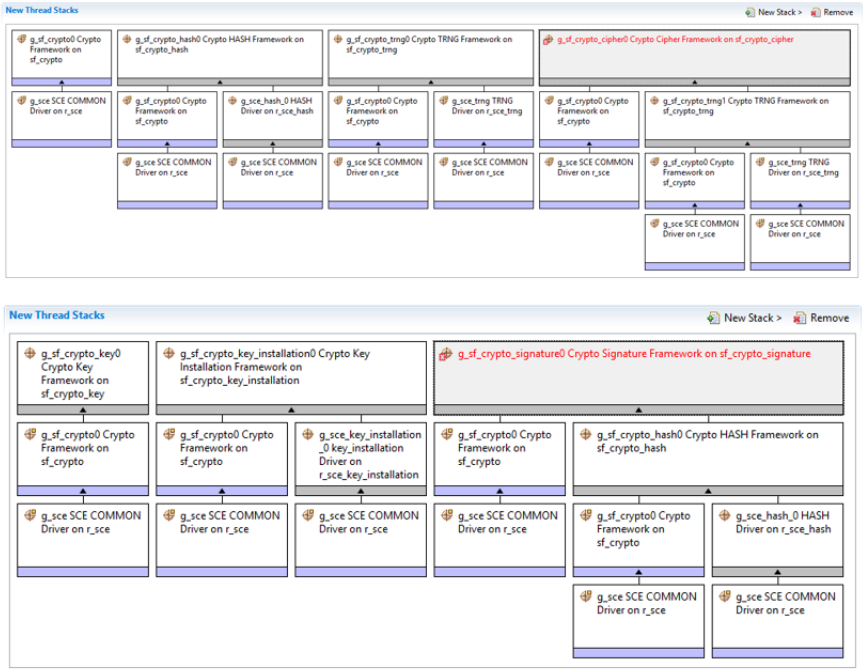


Figure 172: Crypto Framework Module Stack

4.1.16.5 Configuring the Crypto Framework Module

The Crypto Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Crypto Framework on sf_crypto

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto0	Module name.
Wait time	TX_WAIT_FOREVER	Value must be a non-negative integer.
Byte Pool Size	128	Specify the size of the byte pool in bytes.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto will be initialized during startup.
Force Closure Support	Default, Force Close Default: Default	Select Force Close to close the crypto module regardless of the status of submodules.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto HASH Framework on sf_crypto_hash

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Instance Name	g_sf_crypto_hash0	Module name.
Name of generated initialization function	sf_crypto_hash_init0	Select the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_hash will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto TRNG Framework on sf_crypto_trng

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto_trng0	Module name.
Name of generated initialization function	sf_crypt_trng_init0	Select the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_trng will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto Cipher Framework on sf_crypto_cipher

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto_cipher0	Module name.
Key type	RSA Plain text, RSA CRT Plain text, RSA Wrapped, AES Wrapped, ECC Plain text, ECC Wrapped Default: RSA Plain text	Select the key type.
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits, ECC 192-bits, ECC 256-bits Default: RSA 2048-bits	Select the key size.
Cipher chaining mode	ECB, CBC, CTR, GCM, XTS Default: ECB	Select the cipher chaining mode.
Name of generated initialization function	sf_crypto_cipher_init0	Select the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_cipher will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto Key Framework on sf_crypto_key

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto_key0	Module name.
Key type	RSA Plain text, RSA CRT Plain text, RSA Wrapped, AES Wrapped, ECC Plain text, ECC Wrapped Default: RSA Plain text	Select the key type. For AES, the byte pool size defined in sf_crypto must be >= 280 bytes.
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits, ECC 192-bits, ECC 224-bits, ECC 256-bits, ECC 384-bits Default: RSA 2048-bits	Select the key size.
Name of generated initialization function	sf_crypto_key_init0	Select the name of the generated initialization function.
Name of Domain Parameter (Applicable only for ECC)	sf_crypto_key_domain_params0	Specify the name of the ECC domain partner.
Name of Generator Point (Applicable only for ECC)	sf_crypto_key_generator_point0	Specify the name of the ECC generator point.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_key_will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto Key Installation Framework on sf_crypto_key_installation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto_key_installation0	Module name.

Key type	Encrypted RSA Key, Encrypted AES Key, Encrypted ECC Key Default: Encrypted RSA Key	Select the key type.
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits, ECC 192-bits, ECC 256-bits Default: RSA 2048-bits	Select the key size.
Name of generated initialization function	sf_crypto_key_installation_init0	Select the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_key_installation will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Crypto Signature Framework on sf_crypto_signature

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if code for parameter checking is to be included in the build.
Name	g_sf_crypto_signature0	Module name.
Key type	RSA Plain text, RSA CRT Plain text, RSA Wrapped Default: RSA Plain text	Select the key type. Byte Pool size in sf_crypto must be > = 1450 bytes.
Key size	RSA 1024-bits, RSA 2048-bits Default: RSA 2048-bits	Select the key size.
Name of generated initialization function	sf_crypto_signature_init0	Select the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_crypto_signature will be initialized during startup.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

4.1.16.6 Using the Crypto Framework Module in an Application

The typical steps in using the SF CRYPTO Framework module in an application are:

To use the SF CRYPTO module.

- Ensure that the configuration parameters (given in the previous table) are set as per the needs of the application.
- Use the sf_crypto_api_t::open API to Initialize the SF_CRYPTO and the SCE HAL module (R_SCE) through the SCE common driver. This is done by Synergy Configurator when the Auto Initialize setting is at default.
- The little endian mode is set by default. It is locked and not configurable.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_api_t::close API to close the Crypto Framework services and the HW SCE.

Using the SF CRYPTO HASH Framework Module in an Application

The typical steps in using the SF CRYPTO HASH Framework module in an application are:

Calling the APIs in this order: sf_crypto_hash_api_t::open -> sf_crypto_hash_api_t::hashInit -> sf_crypto_hash_api_t::hashUpdate -> sf_crypto_hash_api_t::hashFinal -> sf_crypto_hash_api_t::close.

Details:

- Add SF_CRYPTO HASH Framework module in the Synergy Configurator. The HASH algorithm for this instance is set to SHA 256 by default.
- The little endian mode is set by default. It is locked and not configurable. (Please refer to note on endianness and data format in the Module Operational Notes.)
- The SF CRYPTO Module has to be first opened. (The Synergy Configurator will do this provided the Auto start option set by default is retained.)
- Set the chosen HASH algorithm (specified in Table 4) as the configuration parameter for the module open API.
- Use the sf_crypto_hash_api_t::open API to initialize the SF CRYPTO HASH Framework module and the SCE HASH HAL module (R_SCE_HASH) through the SCE HASH driver. (The Synergy Configurator will do this provided the Auto init option set by default is retained.)
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_hash_api_t::hashInit API to initialize the HASH operation for the chosen HASH algorithm.
    - The sf_crypto_hash_api_t::hashInit can be called after the sf_crypto_hash_api_t::open API, sf_crypto_hash_api_t::hashUpdate API or sf_crypto_hash_api_t::hashFinal API to initialize a new operation with the configured HASH algorithm.
- Use the sf_crypto_hash_api_t::hashUpdate API to hash input data. It can be called multiple times until all the data is completely used up.
- Use the sf_crypto_hash_api_t::hashFinal API to get the message digest for all the data hashed with the sf_crypto_hash_api_t::hashUpdate API.
    - Once the sf_crypto_hash_api_t::hashFinal API is called, either the sf_crypto_hash_api_t::hashInit can be called to initialize operation for the next set of data using the same HASH algorithm or the sf_crypto_hash_api_t::close API can be called to close the module. It can then be re-opened for a different HASH algorithm.
- Use the sf_crypto_hash_api_t::close API to close the Crypto HASH Framework services.

Note

In a particular thread, a single instance of the SF CRYPTO HASH module can be re-used for different HASH algorithms without having to create multiple instances.

Using the SF CRYPTO TRNG Framework Module in an Application

The typical steps in using the SF CRYPTO TRNG Framework module in an application are:

- Use the sf_crypto_trng_api_t::open API to initialize the SF_CRYPTO_TRNG and the SCE TRNG HAL module (R_SCE_TRNG). This is done by the ISDE when the auto initialization setting is set to default.
- The little endian mode is set by default. It is locked and not configurable. (Please refer to the note on endianness and data format in the module Operational Notes.)
- The open function cannot be called again until the module is closed.
- Use sf_crypto_trng_api_t::randomNumberGenerate API to generate random numbers. The minimum length of random number bytes that can be requested is 1.
- Use sf_crypto_trng_api_t::close API to close the Crypto Framework services and the SCE when the TRNG services are no longer required.

Note

In the rare event that the sf_crypto_trng_api_t::randomNumberGenerate API returns an error, all the CRYPTO Framework modules have to closed and SF CRYPTO module re-opened.
There will be only one instance of the TRNG HAL module, so the configuration parameters set for each instance of the Crypto TRNG Framework will be overwrite any previously configured instances. The last value configured is applicable to all instances.

Using the SF CRYPTO Cipher Framework Module in an Application

The typical steps in using the SF CRYPTO Cipher Framework module in an application are:

Calling the APIs in this order: sf_crypto_cipher_api_t::open -> sf_crypto_cipher_api_t::cipherInit -> sf_crypto_cipher_api_t::cipherUpdate -> sf_crypto_cipher_api_t::cipherFinal -> sf_crypto_cipher_api_t::close.

Details:

- Please refer to the note on endianness and data format in the module Operational Notes.
- The Crypto Framework module has to be opened first; the ISDE will do this provided the auto initialization option set by default is retained. Refer to the section regarding using the Crypto Framework module in an Application.
- Set the configuration parameters for the module sf_crypto_cipher_api_t::open API per the needs of the application.
- Use the sf_crypto_cipher_api_t::open API to open the Crypto Cipher Framework module (This is done by the ISDE when the auto start option is set to the default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_cipher_api_t::cipherInit API to initialize the context by assigning appropriate operation mode, key and padding scheme option and algorithm specific parameters.
- If the data for encrypt / decrypt operation is not available all at once but is available in chunks, use the sf_crypto_cipher_api_t::cipherUpdate API. The sf_crypto_cipher_api_t::cipherUpdate API can be called multiple times.

Note

For RSA operations, no data is outputted from the sf_crypto_cipher_api_t::cipherUpdate API. The message will only be accumulated until the sf_crypto_cipher_api_t::cipherFinal API is called.

- In order to complete the encrypt or decrypt operation call the sf_crypto_cipher_api_t::cipherFinal API. This API accepts last chunk of incoming data. In case

all the data has been passed through the sf_crypto_cipher_api_t::cipherUpdate APIs the length for last message chunk can be set to 0.
- In case all the data to be encrypted/decrypted is available at once, sf_crypto_cipher_api_t::cipherFinal API can be called directly without using the sf_crypto_cipher_api_t::cipherUpdate API.
- Use sf_crypto_cipher_api_t::close API to close the Crypto Cipher Framework Module services.

AES GCM operation specifics:

- For AES GCM encrypt / decrypt operations, AAD (Additional Authenticated Data) is optional. If it is to be used, it has to be provided for the cipher operation through the sf_crypto_cipher_api_t::cipherAadUpdate API after the sf_crypto_cipher_api_t::cipherInit is called and prior to calling the sf_crypto_cipher_api_t::cipherUpdate or sf_crypto_cipher_api_t::cipherFinal APIs.
- For AES GCM encrypt operation, the tag (Authentication Tag) will be generated when cipherFinal API is executed. The caller has to supply the buffer to hold the tag through the cipherInit API.
- For AES GCM decrypt operation, the tag has to be provided prior to providing any ciphertext data through the cipherUpdate / cipherFinal APIs. Hence the caller is required to provide the tag through the cipherInit API.
- The AES GCM decrypt operation may output plain text data through the cipherUpdate / cipherFinal APIs but it should not be consumed if the decrypt operation returns an error code. This is to ensure that the tag is verified before the data is used.

Note

In a particular thread, a single instance of the Crypto Cipher Framework module can be re-used for alternately performing encrypt or decrypt operations by appropriately setting/re-setting the operation mode and other related parameters. Two instances of the Crypto Cipher Framework module can also be used simultaneously.

Using the SF CRYPTO KEY Framework Module in an Application

The typical steps in using the SF CRYPTO KEY Framework module in an application are:

Calling the APIs in this order: sf_crypto_key_api_t::open -> sf_crypto_key_api_t::keyGenerate -> sf_crypto_key_api_t::EcdhSharedSecretCompute -> sf_crypto_key_api_t::close.

Details:

- Please refer to note on endianness and data format in the module Operational Notes.
- The Crypto Key Framework has to be opened first; the ISDE will do this provided the auto initialization option is set to the default setting. Refer to the Using the Crypto Framework Module in an Application section.
- Set the configuration parameters as per the needs of the application required for the module sf_crypto_key_api_t::open API.
- Use the sf_crypto_key_api_t::open API to initialize the Crypto Key Framework module; this is done by the ISDE when the auto start option is set to the default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_key_api_t::keyGenerate API to generate the cryptographic keys of the type and size set by the configuration parameters at the open call.
- Use sf_crypto_key_api_t::EcdhSharedSecretCompute API to perform scalar multiplication for ECC algorithms only.
- Use sf_crypto_key_api_t::close API to close the Crypto Key Framework Module services.

Note

In a particular thread, a single instance of Crypto Key Framework module can be re-used for generating different key types without having to create multiple instances.

Using the SF Crypto Key Installation Framework Module in an Application

The typical steps in using the SF Crypto Key Installation Framework module in an application are:

Calling the APIs in this order: sf_crypto_key_installation_api_t::open -> sf_crypto_key_installation_api_t::keyInstall -> sf_crypto_key_installation_api_t::close.

Details:

- Please refer to note on endianness and data format in the module Operational Notes.
- The Crypto Framework Module has to be opened first; the ISDE will do this provided the auto initialization option is set to the default setting. Refer to the Using the Crypto Framework Module in an Application section.
- Set the configuration parameters for the module sf_crypto_key_installation_api_t::open API per the needs of the application.
- Use the sf_crypto_key_installation_api_t::open API to open the Crypto Key Installation Framework module (This is done by the ISDE when the auto start option is set to the default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_key_installation_api_t::keyInstall API to install the user's key.
- Use sf_crypto_key_installation_api_t::close API to close the Crypto Key Installation Framework Module services.

Using the SF CRYPTO Signature Framework Module in an Application

The typical steps in using the SF CRYPTO Signature Framework module in an application are:

Calling the APIs in this order: sf_crypto_signature_api_t::open -> sf_crypto_signature_api_t::contextInit -> sf_crypto_signature_api_t::signUpdate -> sf_crypto_signature_api_t::signFinal -> sf_crypto_signature_api_t::close.

Calling the APIs in this order: sf_crypto_signature_api_t::open -> sf_crypto_signature_api_t::contextInit -> sf_crypto_signature_api_t::verifyUpdate -> sf_crypto_signature_api_t::verifyFinal -> sf_crypto_signature_api_t::close.

Details:

- Please refer to note on endianness and data format in the module Operational Notes.
- The Crypto Framework module has to be opened first; the ISDE will do this provided the auto initialization option is set to the default setting. Refer to the Using the Crypto Framework Module in an Application section.
- Set the configuration parameters per the needs of the application required for the module sf_crypto_signature_api_t::open API.
- Use the sf_crypto_signature_api_t::open API to open the Crypto Signature Framework module (This is done by ISDE when the auto start option is set to the default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the sf_crypto_signature_api_t::contextInit API to initialize the context by assigning appropriate operation mode, key (public key for verify operation mode and private key for sign operation mode), message formatting option.
- If the data is coming in as smaller chunks for sign operation mode, use the

sf_crypto_signature_api_t::signUpdate API to accumulate the data for future signing operation or for verify operation mode use the sf_crypto_signature_api_t::verifyFinal API to accumulate the data for future signature verification.
- In order to complete the sign or verify operation call the sf_crypto_signature_api_t::signFinal or sf_crypto_signature_api_t::verifyFinal APIs, respectively. These APIs support accepting the last chunk of incoming data. In case all the data has been passed through, one of the update API parameters for last message chunk can be set to NULL.
- In case all the data to be signed or verified is available at once, sf_crypto_signature_api_t::signFinal or sf_crypto_signature_api_t::verifyFinal can be called directly without using either of the update APIs.
- Use sf_crypto_signature_api_t::close API to close the Crypto Signature Framework module services.

Note

In a particular thread, a single instance of the Crypto Signature Framework module can be re-used for alternately performing sign or verify operations by appropriately setting/re-setting the operation mode and other related parameters. In order to perform sign or verify operations simultaneously, two instances of the SF Crypto Signature Framework module need to be used.

4.1.17 Capacitive Touch v2 Framework

4.1.17.1 Capacitive Touch v2 Module Introduction

The Capacitive Touch v2 Framework uses the CTSU v2 Driver API and provides application-level APIs for scanning touch buttons, sliders, and wheels. This module is configured via the QE for Capacitive Touch.

4.1.17.2 Capacitive Touch v2 Module Features

- Supports touch buttons (Self and Mutual), sliders, and wheels
- Can retrieve the status of up to 64 buttons at once
- Software and external triggering
- Callback on scan end
- Collects and calculates usable scan results:
    - Slider position from 0 to 100 (percent)
    - Wheel position from 0 to 359 (degrees)
- Optional (build time) support for real-time monitoring functionality through the QE tool over UART

4.1.17.3 Capacitive Touch v2 Module Configuration

Note

This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".
Multiple configurations can be defined within a single project allowing for different scan procedures or button layouts.

The following build time configurations are defined in ssp_cfg/framework/sf_touch_ctsuv2_cfg.h:

Build Time Configurations for sf_touch_ctsuv2

Configuration	Options	Default	Description
Parameter Checking	- Default (BSP) - Enabled - Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support for QE monitoring using UART	- Enabled - Disabled	Disabled	Enable SCI_UART support for QE monitoring.

This module can be added to the Stacks tab by selecting **New Stack > Framework > Input > Cap Touch Framework on sf_touch_ctsuv2**.

Capacitive Touch v2 Module Interrupt Configuration

Refer to the CTSU v2 Driver section for details.

Capacitive Touch v2 Module Clock Configuration

Refer to the CTSU v2 Driver section for details.

Capacitive Touch v2 Module Pin Configuration

Refer to the CTSU v2 Driver section for details.

4.1.17.4 Capacitive Touch v2 Module Usage Notes

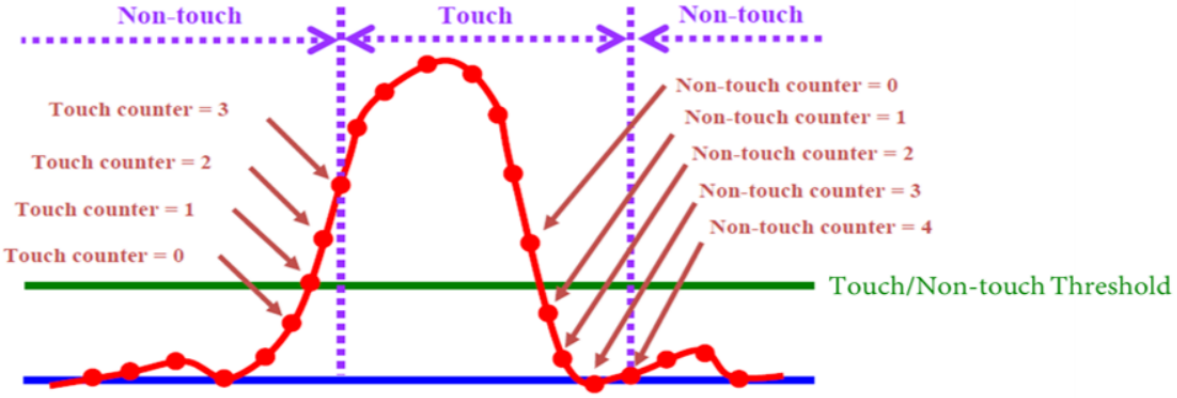
Capacitive Touch v2 Module Sliders and Wheels

Sliders and wheels are subject to some limitations:

	Slider	Wheel
Electrode type	Self capacitance only	Self capacitance only
Number of electrodes	3 to 10	4 or 8
Touch position output range	0-100	0-359
Default value (no touch)	0xFFFF	0xFFFF

Capacitive Touch v2 Module Touch Judgement

Touch data is judged as touched or not-touched based on the threshold and hysteresis values determined during the QE tool tuning process. Refer to the QE for Capacitive Touch tool documentation in e2 studio Help for details on how these values are set.



Measurement Count	Initialization	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7	n+8	n+9	n+10	n+11
User-set touch/non-touch	Non-touch			Touch					Non-touch				
Touch counter buffer	0	0	0	1	2	3	3	3	0	0	0	0	0
Non-touch counter buffer	0	1	2	0	0	0	0	0	1	2	3	4	4
Touch/Non-touch judgement buffer	0	0	0	0	0	1	1	1	1	1	1	0	0
Actual touch/Non-touch judgement	Non-touch					Touch						Non-touch	

Figure 173: Touch/Non-touch judgement Image

4.1.17.5 Capacitive Touch v2 Module Examples

Capacitive Touch v2 Module Basic Example

This is a basic example of minimal use of the TOUCH in an application.

```
void touch_basic_example (void)
{
  ssp_err_t err = SSP_SUCCESS;

  err = SF_TOUCH_CTSU_Open(&g_touch_ctrl, &g_touch_cfg);

  /* Handle any errors. This function should be defined by the user. */
  handle_error(err);
  while (true)
  {
  SF_TOUCH_CTSU_ScanStart(&g_touch_ctrl);
  while (0 == g_flag)
  {
  /* Wait scan end callback */
  }
```

```
  g_flag = 0;
  err = SF_TOUCH_CTSU_DataGet(&g_touch_ctrl, &button, slider, wheel);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
  }
}
```

Capacitive Touch v2 Module Multi-Mode Example

This is a optional example of using both Self-capacitance and Mutual-capacitance. Refer to the Multi Mode Example in CTSUV2 usage notes.

```
void touch_optional_example (void)
{
  ssp_err_t err = SSP_SUCCESS;
  err = SF_TOUCH_CTSU_Open(&g_touch_ctrl, &g_touch_cfg);
  handle_error(err);
  err = SF_TOUCH_CTSU_Open(&g_touch_ctrl_mutual, &g_touch_cfg_mutual);
  handle_error(err);
  while (true)
  {
  SF_TOUCH_CTSU_ScanStart(&g_touch_ctrl);
  while (0 == g_flag)
  {
  /* Wait scan end callback */
  }
  g_flag = 0;
  SF_TOUCH_CTSU_ScanStart(&g_touch_ctrl_mutual);
  while (0 == g_flag)
  {
  /* Wait scan end callback */
  }
```

```
  g_flag = 0;
  err = SF_TOUCH_CTSU_DataGet(&g_touch_ctrl, &button, slider, wheel);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
  err = SF_TOUCH_CTSU_DataGet(&g_touch_ctrl_mutual, &button, slider, wheel);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
  }
}
```

4.1.18 External IRQ Framework

4.1.18.1 External IRQ Framework Module Introduction

The External IRQ Framework provides a high-level API for applications using the external pin interrupts with the ThreadX RTOS and supports the external IRQ pins on the Synergy microcontroller. A callback function (sf_external_irq_callback) is available that will be called from the interrupt service routine (ISR) each time the IRQn triggers.

External IRQ Framework Module Features

- Responds to external interrupt inputs
- RTOS aware implementation using an internal semaphore for thread synchronization
    - Can signal internal threads
    - Can trigger transfers via the Event Link Controller (ELC)
- Uses the port pins available on Synergy MCUs
    - Pins may differ between MCUs so refer to MCU User's Manuals for specifics
- Supports several hardware features such as
    - Channel selection
    - Trigger conditions
    - Digital filtering
    - Auto-start

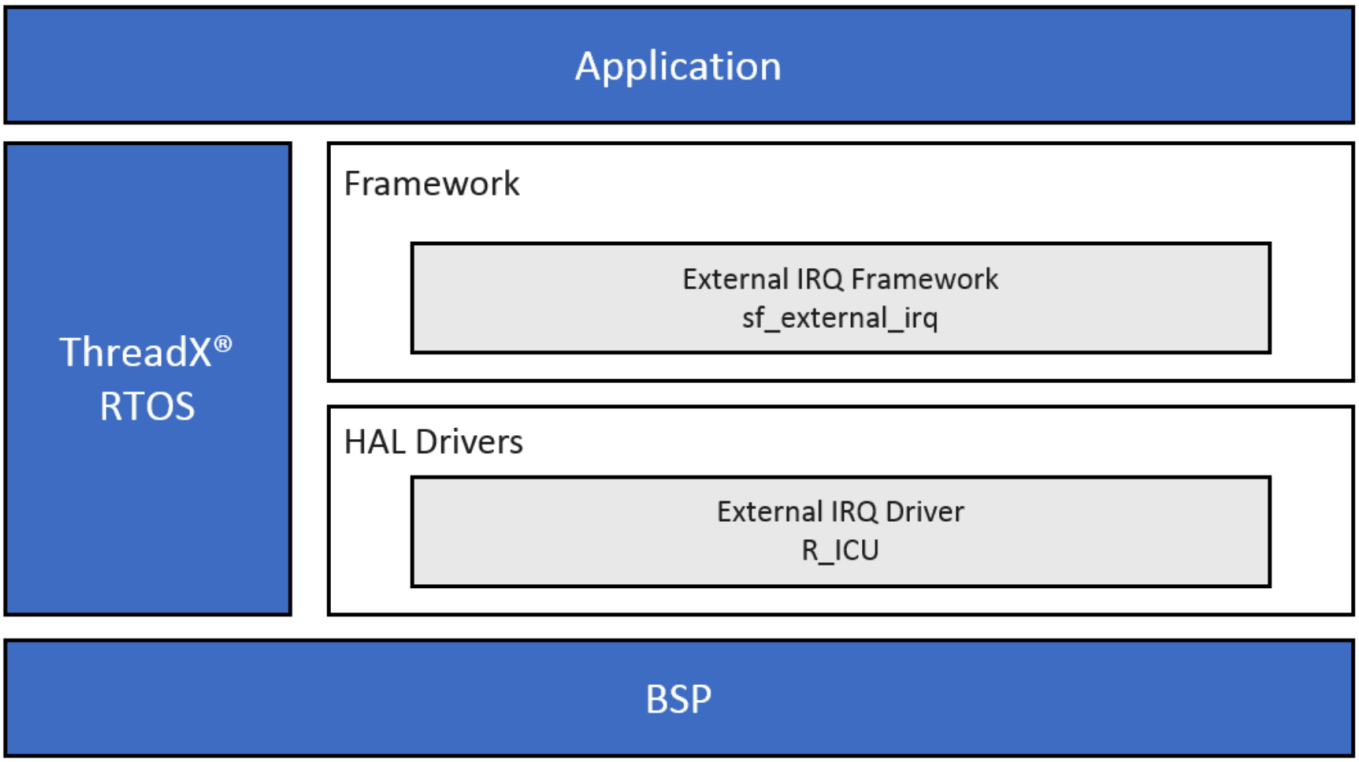


Figure 174: External IRQ Framework Module Block Diagram

4.1.18.2 External IRQ Framework Module APIs Overview

The External IRQ framework module defines APIs for opening, waiting or closing the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

External IRQ Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_external_irq.p_api->open(g_sf_external_irq.p_ctrl, g_sf_external_irq.p_cfg); Acquire mutex, then handle driver initialization at the HAL layer.
wait	g_sf_external_irq.p_api->wait(g_sf_external_irq.p_ctrl, TX_WAIT_FOREVER); Wait for the next external interrupt expiration, then return.
versionGet	g_sf_external_irq.p_api->versionGet(&version); Retrieve the API version and store it in the version pointer.
close	g_sf_external_irq.p_api->close(g_sf_external_irq.p_ctrl); Release channel mutex and close channel at HAL layer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	Device in use.
SSP_ERR_NOT_OPEN	Device unopened.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_WAIT_ABORTED	Suspension aborted.
SSP_ERR_UNSUPPORTED	Function unsupported by the HAL driver.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.18.3 External IRQ Framework Module Operational Overview

The External IRQ framework is a set of ThreadX-aware framework APIs. The External IRQ Framework external inputs can signal, via an internal semaphore, threads or trigger transfers via the Event Link Controller (ELC). Both the External IRQ framework module and the External IRQ HAL module need to be configured for proper operation. The HAL configuration settings allow control over hardware options such as triggering level and digital filtering settings.

External IRQ Framework Module Important Operational Notes and Limitations

External IRQ Framework Module Operational Notes

- Refer to the Datasheet for the Synergy device to be programmed to find the port pins which support the external interrupt functions and to obtain the External IRQ number for a given port pin.
- The External IRQ number corresponds to the channel setting in the ISDE Properties window for the External IRQ driver.

External IRQ Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.18.4 Including the External IRQ Framework Module in an Application

This section describes how to include the External IRQ Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and

configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the External IRQ Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the External IRQ Framework is g_sf_external_irq0. This name can be changed in the associated Properties window.)

External IRQ Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_external_irq0 External IRQ Framework on sf_external_irq	Threads	**New Stack> Framework> Input> External IRQ Framework on sf_external_irq**

When the External IRQ Framework on sf_external_irq is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

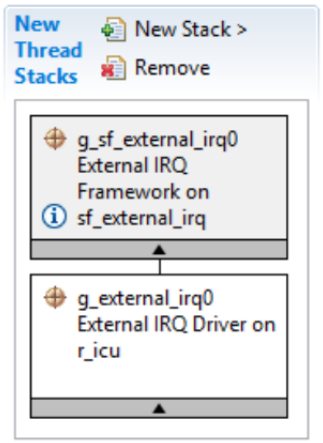


Figure 175: External IRQ Framework Module Stack

4.1.18.5 Configuring the External IRQ Framework Module

The External IRQ Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the External IRQ Framework Module on sf_external_irq

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_sf_external_irq0	Framework name.
Event	None, Semaphore Put Default: Semaphore Put	Event selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling, Rising, Both Edges, Low Level Default: Rising	Configures edge or level triggering.
Digital Filtering	Enabled, Disabled Default: Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64 Default: PCKL/64	Sets noise filter sampling period.

Interrupt enabled after initialization	True, False Default: True	Determines if the interrupt is enabled immediately after initialization.
Callback	NULL	A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	An Interrupt priority can be registered in external_irq_cfg_t::irq_ipl.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

External IRQ Framework Module Clock Configuration

The External IRQ Framework does not require clock configuration.

External IRQ Framework Module Pin Configuration

The External IRQ Framework does not require pin configuration.

4.1.18.6 Using the External IRQ Framework Module in an Application

The steps in using the External IRQ Framework module in a typical application are:

1. Open the External IRQ Framework module with the sf_external_irq_api_t::open API.
2. Wait for an interrupt using the sf_external_irq_api_t::wait API.
3. Process External IRQ event.
4. Close the module using the sf_external_irq_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

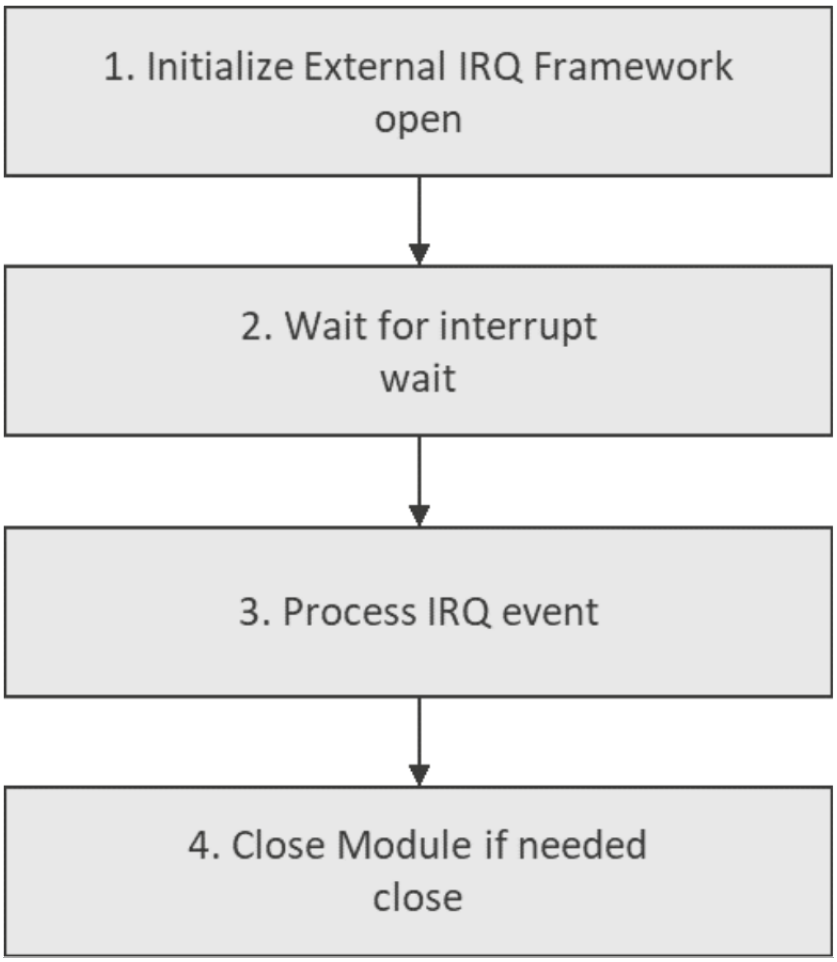


Figure 176: Flow Diagram of a Typical External IRQ Framework Module Application

4.1.19 I2C Framework

4.1.19.1 I2C Framework Introduction

The I2C Framework module provides a ThreadX-aware high-level API for I2C industry standard serial device communications and configures the I2C peripheral in order to enable serial communication to be used by the framework. The I2C Framework module uses the I2C and SCI peripherals on the Synergy MCU.

I2C Framework Module Features

- ThreadX-aware framework
- Handles integration and synchronization of multiple I2C peripherals on the I2C bus
- Provides a single interface to access both SCI I2C and RIIC drivers
- The I2C framework module configures I2C communication in master mode
- The I2C framework module supports three data rates: 100 kHz, 400 kHz, and 1 MHz
- The I2C framework module supports both 7-bit addressing and 10-bit addressing
- The I2C framework module also provides support for callbacks internally. User defined

callback is not used. The callback functions are called with the following events i2c_event_t
    - Transfer aborted
    - Transmit complete
    - Receive complete
- The callback structure i2c_callback_args_t also provides the number of bytes that were sent or received
- Implemented by:
    - Simple I2C on SCI
    - RIIC

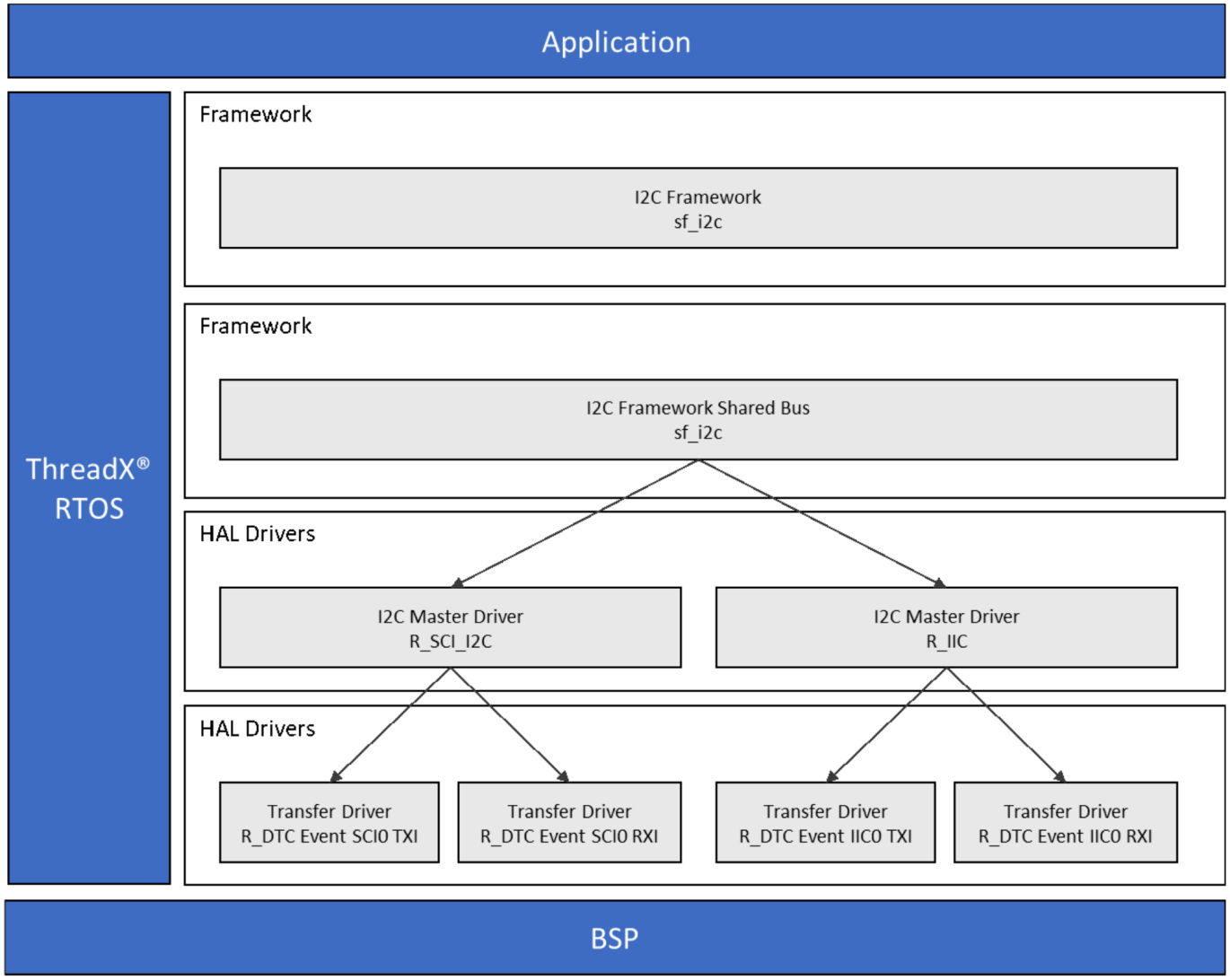


Figure 177: I2C Framework Module Block Diagram

4.1.19.2 I2C Framework Module APIs Overview

The I2C Framework interface defines APIs for opening, closing, reading, writing, locking, unlocking and resetting the bus using the I2C Framework. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2C Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_i2c_device.p_api->open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg); Opens a designated I2C device on the bus.
close	g_sf_i2c_device.p_api->close (g_sf_i2c_device.p_ctrl); Disables I2C device designated by control handle. Closes the RTOS services used by the bus if no devices are connected to the bus.
read	g_sf_i2c_device.p_api->read (g_sf_i2c_device.p_ctrl, &destination, no_of_bytes_to_read, restart,timeout); Receives data from I2C device.
write	g_sf_i2c_device.p_api->write (g_sf_i2c_device.p_ctrl, &source, no_of_bytes_to_write , restart, timeout); Transmits data to I2C device
lock	g_sf_i2c_device.p_api->lock (g_sf_i2c_device.p_ctrl); Locks the bus for a device. Locking reserves the bus until unlocking and allows several reads and writes without interrupt.
unlock	g_sf_i2c_device.p_api->unlock (g_sf_i2c_device.p_ctrl); Unlocks the bus from a particular device and makes it available for other devices.
reset	g_sf_i2c_device.p_api->reset (g_sf_i2c_device.p_ctrl, timeout); Aborts any in-progress transfer and forces the I2C peripheral into ready state.
version	g_sf_i2c_device.p_api->version(version); Retrieves the version information using the version pointer.
lockWait	g_sf_i2c_device.p_api->lockWait (g_sf_i2c_device.p_ctrl, timeout); Locks the bus for a device. Locking reserves the bus until unlocking and allows several reads and writes without intervention from other devices on the same I2C bus. Timeout value is user configurable.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	I2C function performed successfully.
SSP_ERR_INVALID_MODE	Illegal I2C mode is specified.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Omitted I2C channel is specified.
SSP_ERR_IN_USE	I2C channel has already been opened.
SSP_ERR_INVALID_ARGUMENT	Argument is not one of the predefined values.
SSP_ERR_INTERNAL	Internal error has occurred.
SSP_ERR_ASSERTION	A critical assertion has failed or Null pointer(s) is (are) given.
SSP_ERR_NOT_OPEN	Device instance not opened.
SSP_ERR_TRANSFER_ABORTED	The data transfer was aborted.
SSP_ERR_INVALID_RATE	The requested rate cannot be set.
SSP_ERR_TIMEOUT	Timeout error occurs.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.19.3 I2C Framework Module Operational Overview

The I2C Framework module complies with the layered driver architecture of the SSP. It uses the lower-level I2C HAL modules to communicate with I2C peripherals and controls the I2C-capable peripherals on a Synergy microcontroller (as configured by a user). With the I2C Framework module, one or more I2C buses can be created and multiple I2C peripherals can be connected to each I2C bus. The I2C Framework module APIs use a ThreadX-Mutex to acquire and release the shared bus for I2C Slave devices. Acquire and release are implemented by sf_i2c_api_t::lock or sf_i2c_api_t::lockWait and sf_i2c_api_t::unlock APIs respectively in the I2C Framework module.

As I2C framework module configures I2C communication in master mode, this allows the user to:

- Initialize the driver
- Read from a slave device
- Write to a slave device
- Reset the MCU I2C peripheral
- Lock the I2C bus
- Unlock the I2C bus

The I2C Framework module works with the Synergy MCU I2C hardware modules; the RIIC and SCI HAL modules. Both I2C modules support the I2C fast-mode with bit rates of up to 400 kHz. The IIC peripheral and the RIIC HAL module support fast-mode plus with 1-MHz bit-rates. The module supports only master mode for both implementations.

Multiple Slave Devices on the Same Bus

The I2C Framework module uses a bus and device on bus architecture. If multiple slaves are connected to the I2C bus, each slave communicates with an associated and separate SF_I2C module instance. Each SF_I2C instance is created in a separate thread. Every slave device is linked to the

bus to which it will be connected and shares the bus with all other slave devices. The user must configure the framework shared-bus and the lower-level I2C HAL layer for each framework device connecting to the bus. The user can add the existing framework shared-bus when configuring multiple devices on the same bus. A common start and stop procedure is used for all I2C data-transfer operations. Only one device is configured to the lower level and the remaining devices perform read or write operations by switching the device address within the framework.

All I2C Framework devices on the same bus must use the same lower-level configuration settings, (for example, the I2C HAL module) except for the slave address and addressing mode. The framework will use the configuration of the first device that it opens in the application to configure the bus; this means that all I2C Framework devices on the same bus must have the same lower-level configuration settings (except for the slave address and addressing mode). If different configurations are used, proper operation cannot be guaranteed.

Bus Locking

The I2C Framework supports bus-locking functionality, meaning the bus can be locked for a given slave peripheral. The locking allows devices to reserve a bus to themselves for the period between the lock and unlock commands. This allows devices to complete several reads and writes on the bus without interruption (which can be required in some situations.)

The I2C bus is locked when sf_i2c_api_t::lock or sf_i2c_api_t::lockWait API is called. This API locks the I2C bus by acquiring the mutex for the thread in which the I2C Framework device is used. Once locked, the I2C bus can only be utilized by the associated device. The other I2C Framework devices or the same I2C Framework device from other threads, cannot acquire the mutex so they will not be able to access the bus. Once the bus is unlocked by calling the sf_i2c_api_t::unlock API from the sf_i2c device that locked it, the mutex will be released and the bus becomes available for other sf_i2c devices. The sf_i2c_api_t::lockWait API is similar to the sf_i2c_api_t::lock API except it provides user an option to set timeout value. The sf_i2c_api_t::lockWait API waits for the specified timeout period if the I2C bus is already locked by another device. In case of the sf_i2c_api_t::lock API, the thread waits forever, if the I2C bus is not released by the other device.

I2C Framework Module Important Operational Notes and Limitations

I2C Framework Module Operational Notes

- The closest possible baud rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used. If a valid clock rate could not be calculated, an error is returned.
- The I2C can trigger the start of other peripherals available from the ELC. See the ELC Module Guide for further information.
- The I2C Framework can support multiple I2C devices on the same bus if the clock rate remains the same for all the devices. That means multiple devices can be opened in the same bus if they are of the same clock rate. If devices have different clock rates, only one device can be opened at a time.
- SDA and SCL output pin type should be n-channel open drain when using I2C on SCI.
- In the I2C Framework configuration, the channel number given to this bus overrides the channel number given in the HAL module.
- Shared bus can be used by multiple slave devices with the respective configuration. The framework also handles mutual exclusion in lock and unlock APIs when multiple devices are using the same I2C channel.
- To configure multiple I2C devices on the same bus, add and configure the following modules for each device connecting to the bus:
    - I2C Framework device module
    - Configure the I2C shared bus module for the first device being configured, then

use the same bus for the remaining devices.
    - I2C HAL module
    - DTC module (Optional)
- Lock functionality will be effective for devices from different threads. If multiple devices connected to the bus are from the same thread, the I2C bus will be locked for all devices from that thread. In such cases, even if the bus is locked, all devices from the same thread can access the bus.
- In case a device is being used from multiple threads, and the device locks the I2C bus from one thread, the same device cannot access the I2C bus from other threads.

Note

Each I2C Framework device must be configured with a unique name in the ISDE configurator.
Provide the same configuration settings for all the devices connected on the same bus (except the slave address and addressing modes.)

I2C Framework Module Limitations

The I2C framework module does not currently support the following feature:

- The use of DMA

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.19.4 Including the I2C Framework Module in an Application

This section describes how to include the I2C Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2C Framework module to an application, simply add it to a HAL/Common thread using the stacks selection sequence given in the following table. (The default name for the I2C Framework module is g_sf_ i2c_device0. This name can be changed in the associated Properties window.)

I2C Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_i2c_device0 I2C Framework on sf_i2c	Threads	**New Stack> Framework> Connectivity> I2C Framework on sf_i2c**

When the I2C Framework module on sf_i2c is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

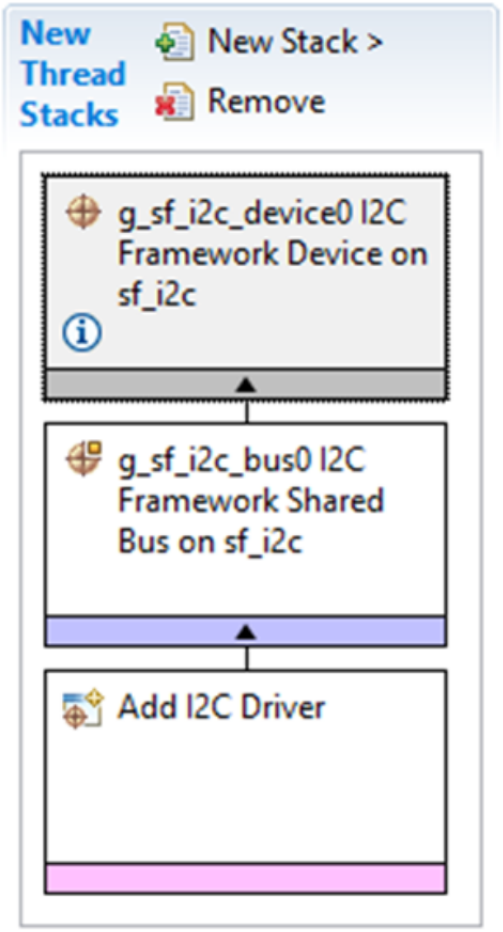


Figure 178: I2C Framework Module Stack

4.1.19.5 Configuring the I2C Framework Module

The I2C Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the I2C Framework Device Module on sf_ i2c

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: Enabled	Selects if code for parameter checking is to be included in the build.
Name	g_sf_i2c_device0	Give a name to identify the I2C Framework device. API, Config and Control instances will be created based on this name.
Slave Address	0x00	Specify the address of the I2C slave device.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Select the I2C address mode.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different byte ordering or pixel-color format. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the I2C Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules:

Configuration Settings for the I2C Framework Shared Bus on sf_i2c

ISDE Property	Value	Description
Name	g_sf_i2c_bus0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master Driver on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_i2c0	Module name.

Channel	0	Specify the IIC channel to be used with this configuration.
Rate	Standard, Fast-mode, Fast-mode Plus Default: Standard	Standard, Fast, and Fast-plus. (See IIC Rate Calculation.)
Slave Address	0x00	Set the address of the slave device the I2C master will be communicating with.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nanoseconds)	Default: 300	SDA output delay in nanoseconds.
Timeout Mode	Short Mode, Long Mode Default: Short Mode	Select the timeout mode.
Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. The exact hardware generated error event is also provided when i2c_event_t is I2C_EVENT_ABORTED. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the receive interrupt priority.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the transmit interrupt priority.

Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the transmit end interrupt priority.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the error interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Fixed	Select the transfer size.
Source Address Mode	Incremented	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode)	Source	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.

Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event IIC0 TXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Incremented	Select the transfer size.
Source Address Mode	Fixed	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Destination	Select the address mode for the source.

Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event IIC0 RXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master Driver on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_i2c0	Module name.
Channel	0 to 9	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7: 0 1 2 3 4 5 6 7 8 9; Series S3 : 0 1 2 3 4 - - - - 9; Series S1 : 0 1 - - - - - - - 9.

Rate	Standard, Fast-mode, Fast-mode plus Default: Standard	Select the I2C data rate.
Slave Address	0x00	Specify the slave address.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.
Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the receive interrupt priority.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit interrupt priority.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit end interrupt priority.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the error interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer0	Module name.
Mode	Block	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Fixed	Select the transfer size.
Source Address Mode	Incremented	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Source	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Incremented	Select the transfer size.
Source Address Mode	Fixed	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Destination	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.

Activation Source (Must enable IRQ)	Event SCI0 RXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I2C Framework Module Clock Configuration

The SCI peripheral module uses the PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator clock tab prior to a build or by using the CGC Interface at run-time. During configuration, the I2C transfer rate is calculated and set internally by the driver based on the user-selected PCLB rate and the user-selected transfer rate. If the PCLKB is configured in such a manner that the user-selected rate cannot be achieved, an error will be returned when initializing the driver.

I2C Framework Module Pin Configuration

The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note

For some peripherals, the operation-mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for the I2C Framework Module

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals**> **SCI1_3_5_7_9**> **SCI1**

Note

The selection sequence assumes the SCI1 is the desired hardware target of the driver.

Pin Configuration Settings for the I2C Framework Module

Pin Configuration Property	Value	Description

Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Simple I2C as the Operation Mode for I2C on SCI.
RXD1_SCL1_MISO1	None, P212, P708 Default: None	SCL pin.
TXD1_SDA1_MOSI1	None, P213, P709 Default: None	SDA pin.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

I2C Framework Module Additional Settings

In addition to the SCL and SDA pins, an I2C RESET signal may be required to reset the I2C slave device. If this is the case, the RESET signal can be added using a GPIO pin and must be controlled directly by the application program. The external device reset function is not supported within the r_sci_i2c module.

4.1.19.6 Using the I2C Framework Module in an Application

A common application for the I2C framework module requires multiple slave devices on a single bus. The implementation for this common application is described below. (For an application where multiple busses are required, just duplicate the single bus example as needed for each separate bus.)

Implementation Steps for Two Slave Devices on the Same Shared Bus

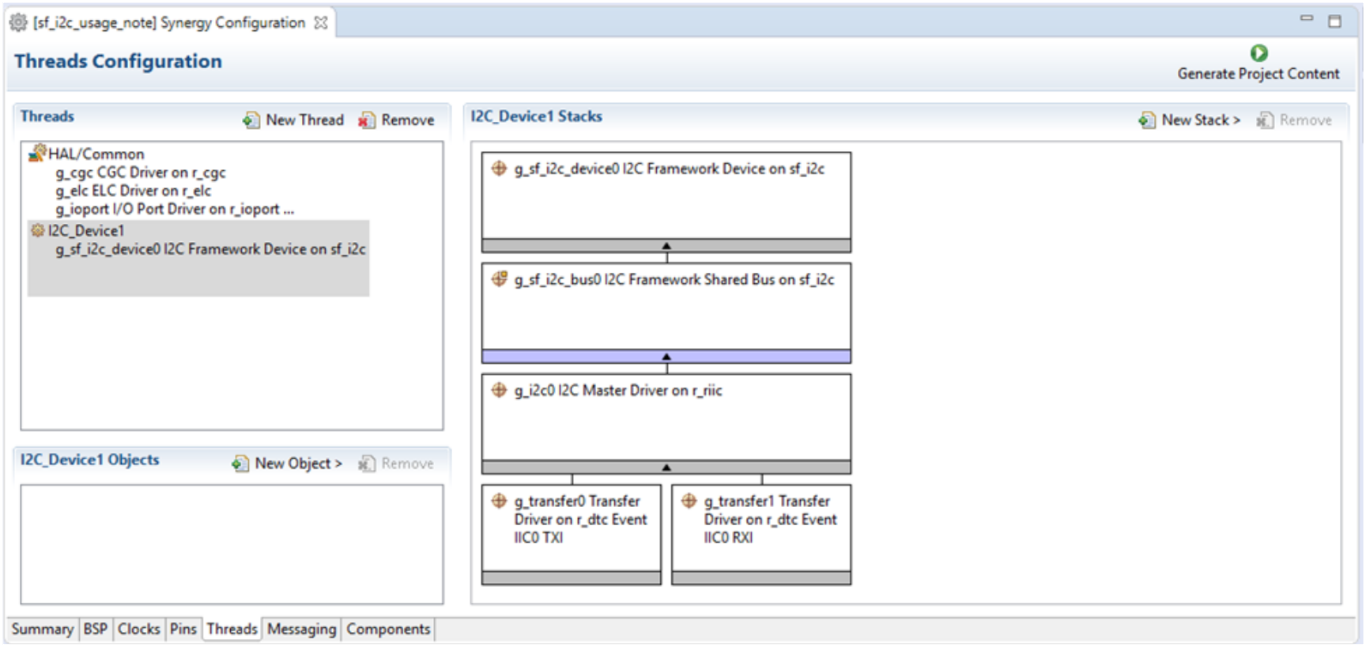
When using the I2C framework module to create a single bus with multiple slave devices, create two thread stacks each with an I2C framework instance. These instances will each use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note

The following example puts both sf_i2c module instances in the same thread. If the bus locking function is needed, the sf_i2C modules should be put in different threads. Locking applies to all the devices within the locked thread. The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.

1. Add the first I2C framework device to a new or existing thread. This creates the I2C master stack. A shared bus on sf_i2c is added along with the I2C driver. The I2C driver can be selected for implementation on r_riic or r_sci_i2c. The DTC transfer driver is also added by default. This can be removed if the CPU transfer mode is needed instead.

The resulting module stack is shown in the following figure. Example configuration settings are given in the tables that follow the figure:



Example configuration setting for the first thread stack are given below:

Configuration Settings for the I2C Framework Device on sf_i2c (Slave #1)

Property	Value	Description
Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.
Name	g_sf_i2c_device0	Give a name to identify the I2C Framework device. API, Config and Control instances will be created based on this name.
Slave Address	0x21	Specify the address of I2C slave 1.
Address Mode	7-bit	Select the I2C address mode.

Configuration Settings for the I2C Framework Shared Bus on sf_i2c

Property	Value	Description
Name	g_sf_i2c_bus0	Give a name to identify the I2C Framework shared bus. This shared bus will be shared by multiple I2C Framework Devices.

Configuration Settings for the I2C Master Driver on r_riic

Property	Value	Description

Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.
Name	g_i2c0	Give a name to identify the I2C Driver device. This will be used internally by Framework.
Channel	0	Specify the I2C channel.
Rate	Standard	Select the speed of the I2C bus.
Slave Address	0	This field will be locked as slave address already configured in the I2C Framework Device on sf_i2c.
Address Mode	7-bit	This field will be locked as address mode already configured in the I2C Framework Device on sf_i2c.
Timeout Mode	Short Mode	Select Timeout mode: Short mode or Long mode.
Callback	NULL	This field will be locked as Framework does not provide callback handling to the user.
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 2	Error interrupt priority selection.

Configuration Settings for the I2C Master Driver on r_sci_i2c

Property	Value	Description
Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.
Name	g_i2c0	Give a name to identify the I2C Driver device. This will be used by Framework internally.
Channel	0	Specify the address of I2C slave.
Rate	Standard	Select the speed of the I2C bus.
Slave Address	0	This field will be locked as slave address already configured in the I2C Framework Device on sf_i2c.

Address Mode	7-bit	This field will be locked as address mode already configured in the I2C Framework Device on sf_i2c.
Slave Output Delay	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable	Enables bitrate modulation function.
Callback	NULL	This field will be locked as Framework does not provide callback handling to the user.
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 TXI

Property	Value	Description
Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.
Software Start	Disabled	
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection. This field is locked.
Transfer Size	1 Byte	Transfer size selection. This field is locked.
Destination Address Mode	Fixed	Destination address mode selection. This field is locked.
Source Address Mode	Incremented	Source address mode selection. This field is locked.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection. This field is locked.
Interrupt Frequency	After all transfers have completed	This field is locked.
Destination Pointer	NULL	Destination pointer selection. This field is locked.
Source Pointer	NULL	Source pointer selection. This field is locked.

Number of Transfers	0	Number of transfer selection. This field is locked.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection. This field is locked.
Activation Source (Must enable IRQ)	Event IIC0 TXI	Activation source selection. This field is locked.
Auto Enable	False	Auto enable selection. This field is locked.
Callback (Only valid with Software start)	NULL	Callback selection. This field is locked.
ELC Software Event Interrupt Priority	Disabled	ELC software event interrupt priority selection.

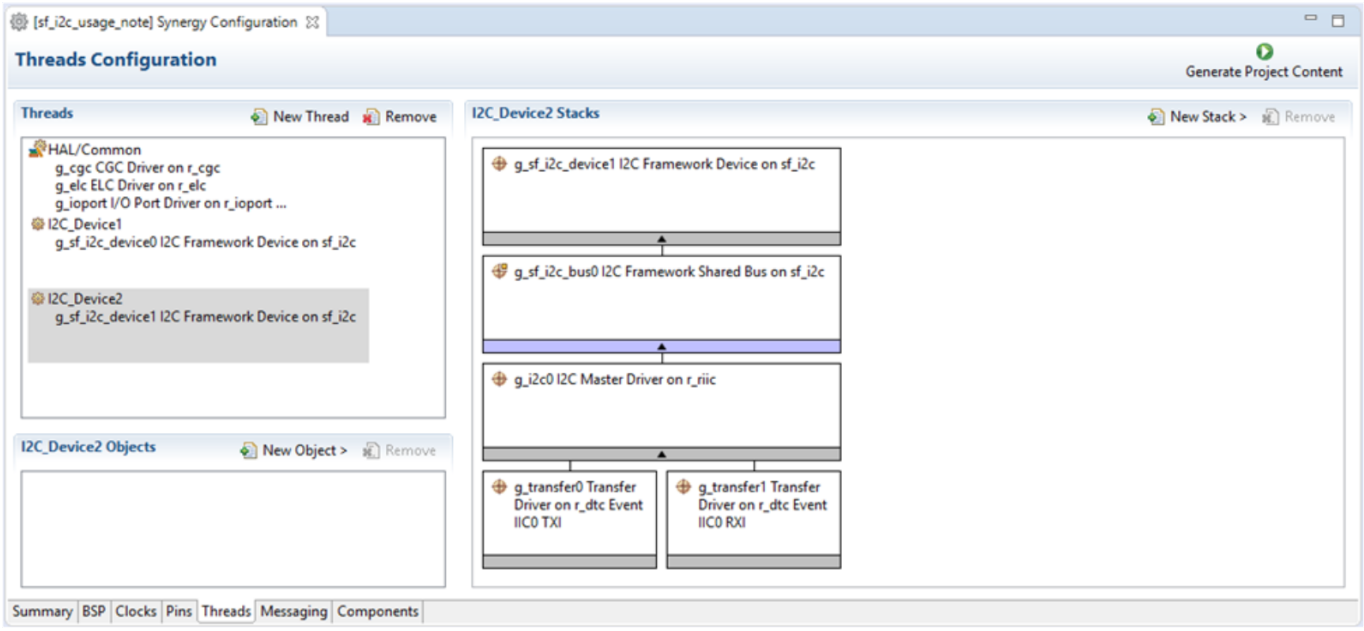
Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

Property	Value	Description
Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.
Software Start	Disabled	Software Start Selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection. This field is locked.
Transfer Size	1 Byte	Transfer size selection. This field is locked.
Destination Address Mode	Incremented	Destination address mode selection. This field is locked.
Source Address Mode	Fixed	Source address mode selection. This field is locked.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection. This field is locked.
Interrupt Frequency	After all transfers have completed	This field is locked.
Destination Pointer	NULL	Destination pointer selection. This field is locked.
Source Pointer	NULL	Source pointer selection. This field is locked.
Number of Transfers	0	Number of transfer selection. This field is locked.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection. This field is locked.

Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection. This field is locked.
Auto Enable	False	Auto enable selection. This field is locked.
Callback (Only valid with Software start)	NULL	Callback selection. This field is locked.
ELC Software Event Interrupt Priority	Disabled	ELC software event interrupt priority selection.

2. Add the second I2C Framework Device to the same thread. The I2C Framework Shared Bus on sf_i2c will not get added automatically. Select the option to use the existing shared bus. Then Configurator will automatically add the I2C Framework Shared Bus on sf_i2c and remaining modules. The lower level modules will be added and configured automatically to be consistent with the previously defined settings from the first I2C framework instance. In fact, if any lower level settings are changed in on stack, they are automatically updated in the other.

The resulting module stack is shown in the following figure:



Example configuration settings for the second thread stack (Slave Device #2) are as follows:

Configuration Settings for the I2C Framework Device on sf_i2c (Slave #2)

Property	Value	Description
Parameter Checking	Default(BSP)	Enable Or Disable Parameter Checking.

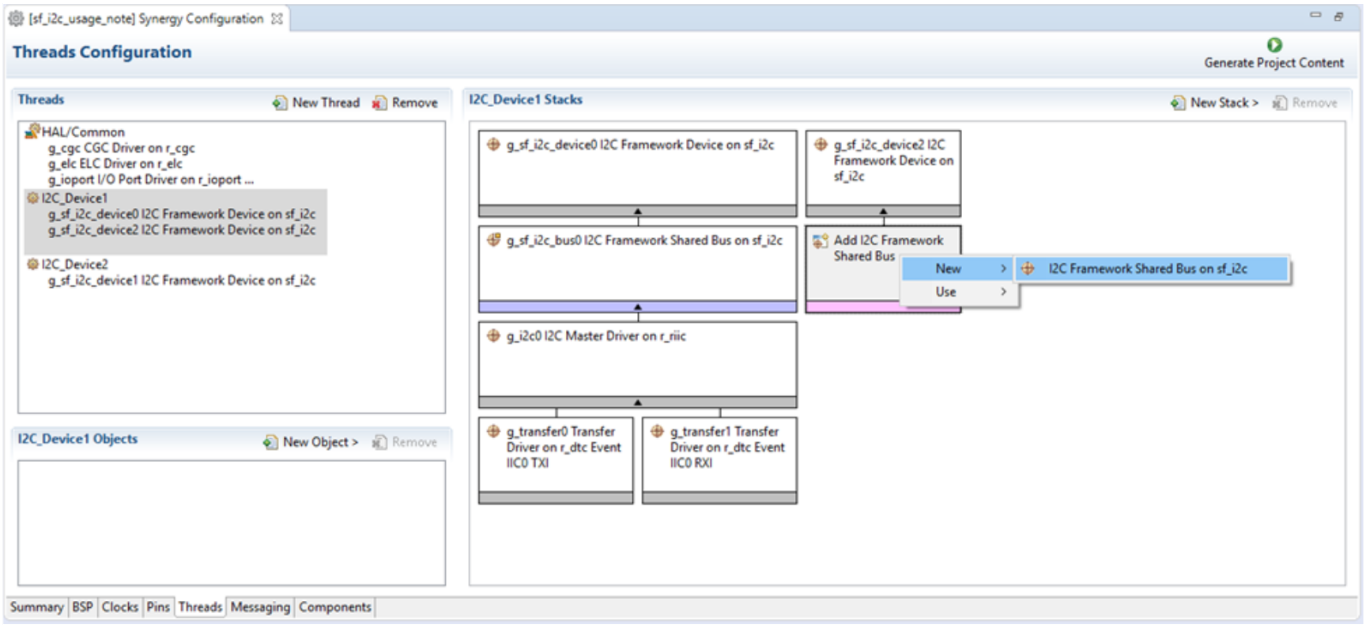
Name	g_sf_i2c_device1	Give a name to identify the I2C Framework device. API, Config and Control instances will be created based on this name.
Slave Address	0x28	Specify the address of I2C slave2.
Address Mode	7-bit	Select the I2C address mode.

The step above can be repeated as needed to add more slave devices to the same bus, if they share the same low-level settings.

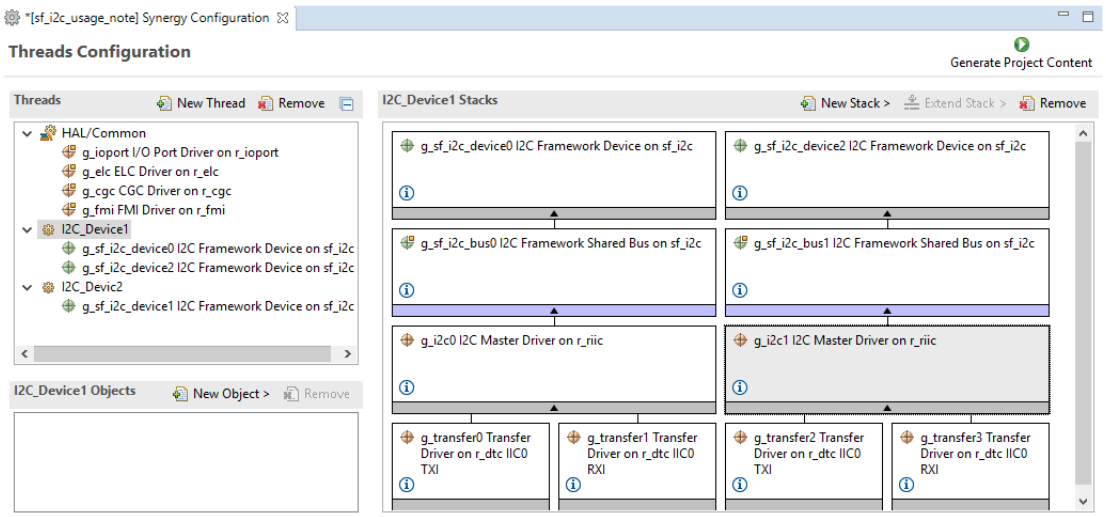
If a set of slave devices have different lower-level settings than another set, they must use a different bus and can be implemented by repeating the two steps outlined above- defining a different bus and the different lower-level characteristics for the set of slave devices.

Adding Another Shared Bus

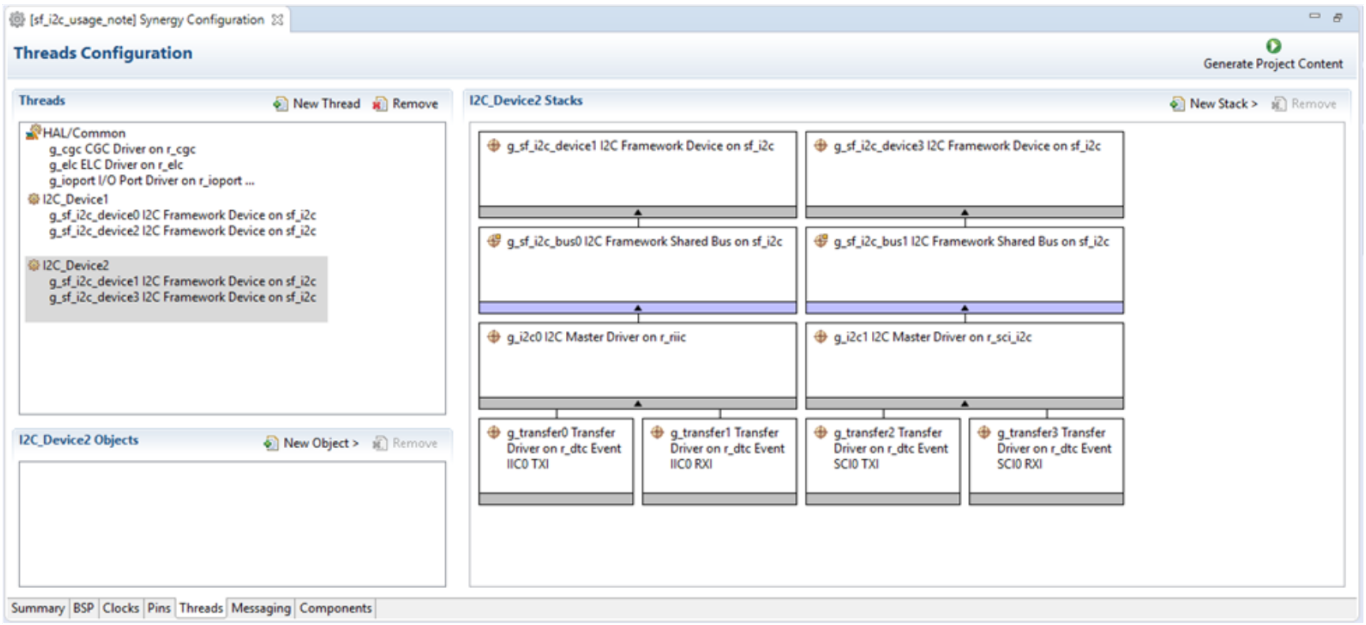
1. The I2C framework module which will use a second shared bus can be added to any thread. Starting with the previous example, if it is added to the I2C_Device1 thread, then the module stack would appear as shown below. Available options for the shared bus are **New** or **Use**.



2. Select **New** to and add another I2C Framework Shared Bus on sf_i2c module. Configure the shared bus properties as needed for the application. Select the desired low-level I2C driver. The channel number for the g_i2c1 I2C driver module, must be different from the channel number for the g_i2c0 I2C driver module. The resulting thread stack is shown below:



3. A second device can be added in the I2C_Device2 thread using the same steps described above. The resulting thread stack is shown below:



The typical steps in using the I2C Framework module in an application are:

1. Initialize the I2C Framework module using the sf_i2c_api_t::open API. Each I2C framework module needs to call sf_i2c_api_t::open API at least once before performing any operations on the bus.
2. Reset the I2C MCU peripheral using the sf_i2c_api_t::reset API (if needed)
3. Lock the bus using the sf_i2c_api_t::lock or sf_i2c_api_t::lockWait API for a particular framework module. Once the bus is locked by an I2C framework module it cannot be used by any other I2C framework module on the same bus. This ensures that ownership of the bus remains with the I2C framework module until it unlocks it. Any operation from other I2C

framework modules on the bus will fail while the bus is locked. It is not mandatory to lock the bus before any read/write operations on the bus. It is optional (if needed). If thread is not supposed to wait forever when locking the I2C bus, call sf_i2c_api_t::lockWait API with desired timeout value.

4. Write data to the slave using the sf_i2c_api_t::write API. The write operation will not be successful if the bus is already locked by any other I2C framework module.
5. Read data from the slave using sf_i2c_api_t::read API. The read operation will not be successful if the bus is already locked by any other I2C framework module.
6. Unlock the bus using the sf_i2c_api_t::unlock API if it is already locked by the same I2C framework module. Once the bus is unlocked other I2C framework modules can use it. It is necessary to unlock the locked bus after the protected read or write operations are over (if needed).
7. Close the I2C framework module using the sf_i2c_api_t::close API. Each I2C framework module can call the sf_i2c_api_t::close API after all its read and write operations on the bus are completed (if needed).

These common steps are illustrated in a typical operational flow in the following figure:

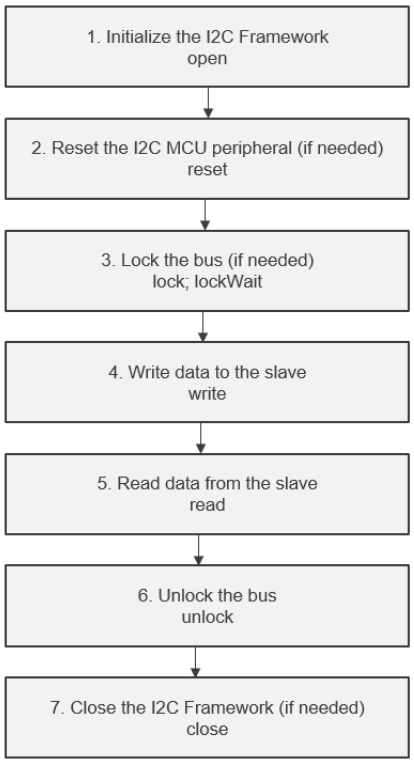


Figure 179: Flow Diagram of a Typical I2C Framework Module Application

4.1.20 JPEG Decode Framework

4.1.20.1 JPEG Decode Framework Module Introduction

The JPEG Decode HAL module provides a high-level API for industry standard JPEG image decode processing and supports the Renesas Synergy™ JPEG Codec peripheral. The JPEG Decode Framework

Module is a ThreadX®-aware implementation and provides thread-safe access to the Synergy JPEG hardware on a Synergy MCU. A user-defined callback can be created to detect hardware supported events.

JPEG Decode Framework Module Features

- Provides thread-safe access to the Synergy JPEG hardware.
- Supports JPEG decompression using the JPEG Decode HAL module.
- Supports a polling mode that allows an application to wait for the JPEG Decoder to complete.
- Supports an interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into the JPEG Decoder module. This feature allows an application to read a coded JPEG image from a file or from a network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded formats YCbCr444, YCbCr422, YCbCr420 and YCbCr411.
- Supports the output formats ARGB8888 and RGB565.
- Returns an error when the JPEG image's size, height, and width do not meet the requirements.
- Supports the sf_jpeg_decode_api_t::wait API function to suspend/resume the thread for synchronizing with the JPEG hardware supported events.

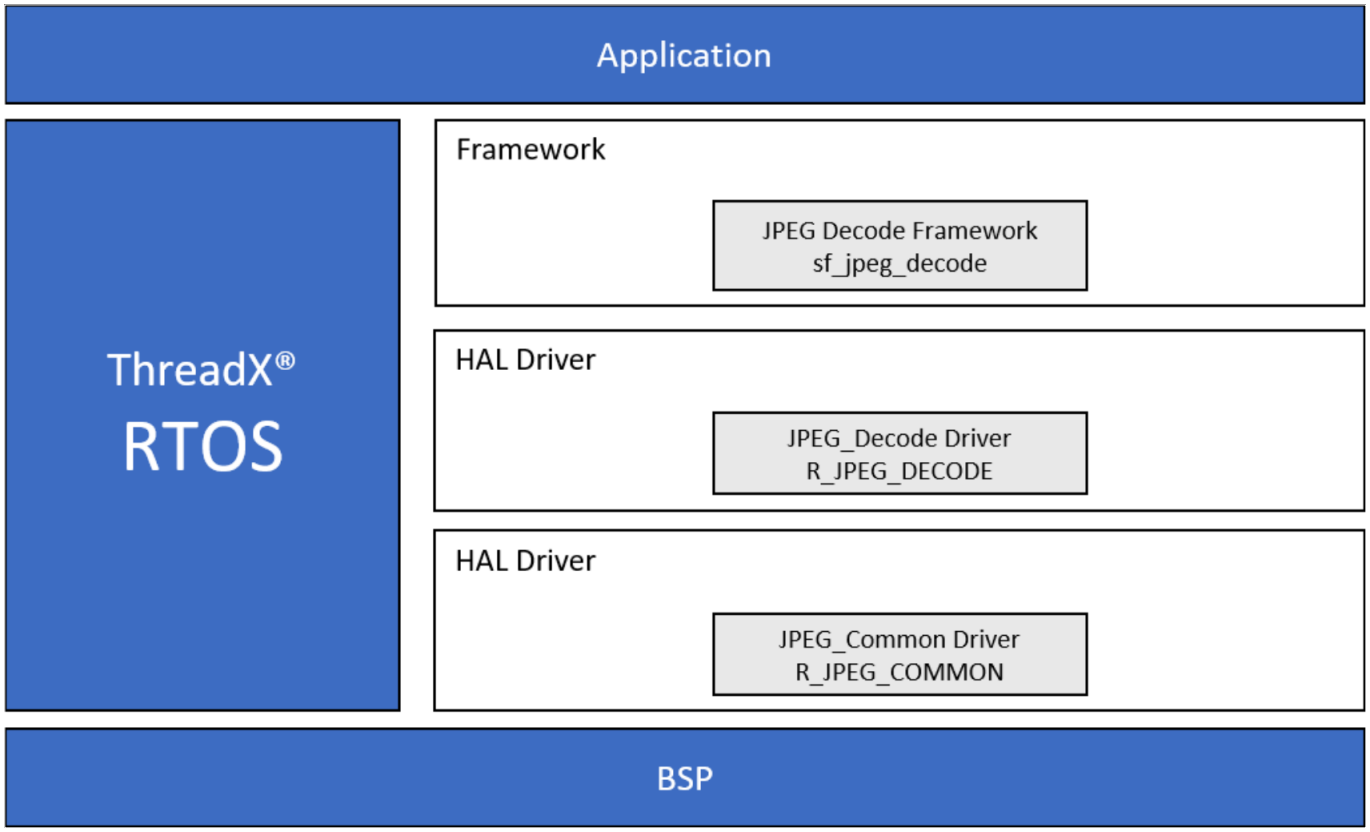


Figure 180: JPEG Decode Framework Module Block Diagram

4.1.20.2 JPEG Decode Framework Module APIs Overview

The JPEG Decode Framework module defines APIs for opening, closing, setting alarms and starting and stopping RTC operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

JPEG Decode Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_jpeg_decode0.p_api->open(g_sf_jpeg_decode0.p_ctrl, g_sf_jpeg_decode0.p_cfg); Open the JPEG Decode Framework.
inputBufferSet	g_sf_jpeg_decode0.p_api->close(g_sf_jpeg_decode0.p_ctrl); Close the JPEG Decode Framework.
outputBufferSet	g_sf_jpeg_decode0.p_api->outputBufferSet(g_sf_jpeg_decode0.p_ctrl, p_buffer, buffer_size); Assign output buffer to JPEG codec for storing output data.
linesDecodedGet	g_sf_jpeg_decode0.p_api->linesDecodedGet g_sf_jpeg_decode0.p_ctrl, p_lines); Return the number of lines decoded into the output buffer.
horizontalStrideSet	g_sf_jpeg_decode0.p_api->horizontalStrideSet(g_sf_jpeg_decode0.p_ctrl, stride); Configure the horizontal stride value.
imageSubsampleSet	g_sf_jpeg_decode0.p_api->imageSubsampleSet(g_sf_jpeg_decode0.p_ctrl, horizontal, vertical); Configure the horizontal and vertical subsample settings.
wait	g_sf_jpeg_decode0.p_api->wait(g_sf_jpeg_decode0.p_ctrl, p_status, timeout); Wait for the current JPEG codec operation to finish with a timeout value given in ThreadX ticks.
statusGet	g_sf_jpeg_decode0.p_api->statusGet(g_sf_jpeg_decode0.p_ctrl, p_status); Retrieve current status of the JPEG codec module.
imageSizeGet	g_sf_jpeg_decode0.p_api->imageSizeGet(g_sf_jpeg_decode0.p_ctrl, p_horizontal, p_vertical); Retrieve image size during decoding operation.
pixelFormatGet	g_sf_jpeg_decode0.p_api->pixelFormatGet(g_sf_jpeg_decode0.p_ctrl, p_color_space); Get the input pixel format.

close	g_sf_jpeg_decode0.p_api->close(g_sf_jpeg_decode0.p_ctrl); Cancel an outstanding operation.
versionGet	g_sf_jpeg_decode0.p_api->versionGet(&version); Get version and store it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	JPEG Decode driver is successfully opened.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	Module already in use.
SSP_ERR_TIMEOUT	The wait operation times out, the underlying driver did not respond in time.
SSP_ERR_WAIT_ABORTED	System internal error occurred.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.20.3 JPEG Decode Framework Module Operational Overview

The JPEG Decode Framework module implements the standard JPEG decode operation. It takes the data in an input buffer and applies the defined JPEG decode algorithm to the buffer, the output is then delivered to the defined output buffer location. A wait API function can be used to suspend/resume the thread for synchronization with JPEG hardware supported events.

JPEG Decode Framework Module Important Operational Notes and Limitations

JPEG Decode Framework Module Operational Notes

- Start decoding JPEG-encoded data by calling the sf_jpeg_decode_api_t::open API. To open the module, use the JPEG Decode Framework module instance, that includes the API function pointer, the pointer to the control block and static configuration that is generated through the Synergy Project configurator in the e^2 studio for ISDE.
- Stop the JPEG Decode Framework module by calling the sf_jpeg_decode_api_t::close API.
- An input buffer-streaming mode is available when an input-centric function is needed.
- An output buffer-streaming mode is available when an output-centric function is needed.
- Supports RGB565 and ARGBB888 output data-color formats.
- The JPEG Decode Framework module has a status flag in the control block, which provides the current status of the module through the sf_jpeg_decode_api_t::statusGet API. The status is also reported through a user-callback function when specific events occur in the module.
- The JPEG Decode Framework module supports buffer-streaming mode for the input buffer in

cases when the input buffer is smaller than the source image size. Set the next input frame as an input buffer every time there is a hardware-generated INPUT_PAUSE interrupt.
- The JPEG Decode Framework module supports buffer-streaming mode for the output buffer in cases when the resultant image is larger than the output buffer-size. Read and store data from the output buffer to make space for upcoming data every time there is a hardware generated OUTPUT_PAUSE interrupt.
- The input and output buffers should be 8-bytes aligned for the JPEG Decode Framework module to be successful. Otherwise, API functions will return error codes that indicate unsuccessful execution.

JPEG Decode Framework Module Limitations

- The JPEG Decode Framework module does not support JPEG-encode processing.
- Check for timeout error (SSP_ERR_TIMEOUT) using sf_jpeg_decode_api_t::wait API; if it returns timeout error, close the framework, re-open it and then perform the decoding operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.20.4 Including the JPEG Decode Framework Module in an Application

This section describes how to include the JPEG Decode Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Decode Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the JPEG Decode Framework is g_sf_jpeg_decode0. This name can be changed in the associated Properties window.)

JPEG Decode Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_jpeg_decode0 JPEG Framework	Threads	**New Stack> Framework> Graphics> JPEG Decode Framework on sf_jpeg_decode**

When the JPEG Decode Framework on sf_jpeg_decode is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

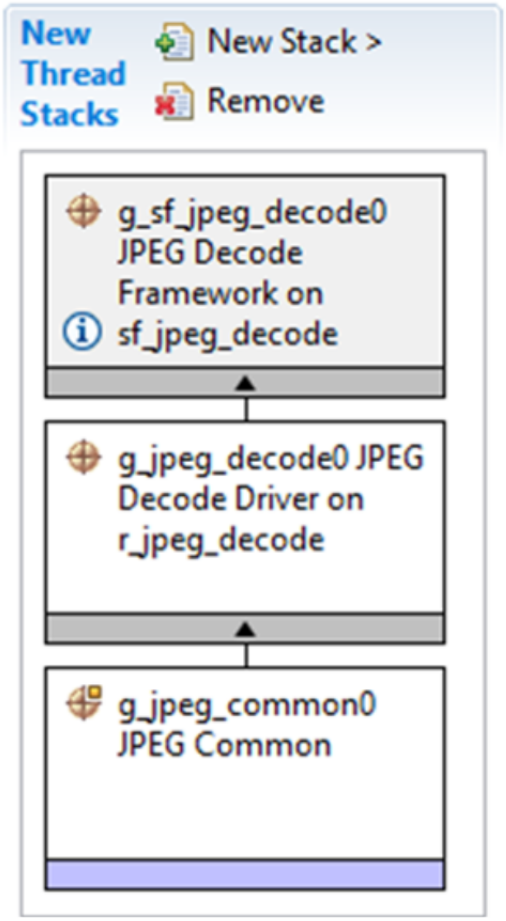


Figure 181: JPEG Decode Framework Module Stack

Decompression Process Interrupt (JEDI)

The JPEG decompression-process interrupt occurs when:

- The current decompression process is successfully completed.
- An error happens in the decompression process.
- Image size and pixel format are successfully read out.

Data Transfer Interrupt (JDTI)

The JPEG data-transfer interrupt occurs when:

- All the JPEG-coded data has successfully completed.
- The number of output image-data lines specified by sf_jpeg_decode_api_t::linesDecodedGet has been transferred.

The number of input image-data lines specified by sf_jpeg_decode_api_t::inputBufferSet has been transferred.

4.1.20.5 Configuring the JPEG Decode Framework Module

The JPEG Decode Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the

properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the JPEG Decode Framework Module on sf_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_jpeg_decode0	The name to be used for a JPEG Decode Framework module instance.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Decode Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the JPEG HAL Module on r_jpeg

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.

Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Decompression interrupt priority selection.

Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Data transfer interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Common Module

ISDE Property	Value	Description
Name	g_jpeg_common0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

JPEG Decode Framework Module Clock Configuration

The JPEG Framework module uses the peripheral module clock A (PCLKA) to run the internal logic.

JPEG Decode Framework Module Interrupt Configuration

To enable interrupts, set the priority of the decompression interrupt and the data-transfer interrupt in the **Properties** window of the JPEG Decode Framework module in the ISDE.

JPEG Decode Framework Module Pin Configuration

The JPEG Decode Framework module does not use any pins.

4.1.20.6 Using the JPEG Decode Framework Module in an Application

The steps in using the JPEG Decode Framework module in a typical application are:

1. Initialize the JPEG Decode peripheral using the sf_jpeg_decode_api_t::open API.
2. Set Image Subsample using the sf_jpeg_decode_api_t::imageSubsampleSet API.
3. Set Horizontal stride using the sf_jpeg_decode_api_t::horizontalStrideSet API.
4. Set output buffer using the sf_jpeg_decode_api_t::outputBufferSet API.
5. Set Input buffer using the sf_jpeg_decode_api_t::inputBufferSet API.
6. Wait for decode to complete with the sf_jpeg_decode_api_t::wait API.
7. Check decode status with the sf_jpeg_decode_api_t::statusGet API.
8. Close the instance with the sf_jpeg_decode_api_t::close API (if needed).

These common steps are illustrated in a typical operational flow diagram in the following figure:

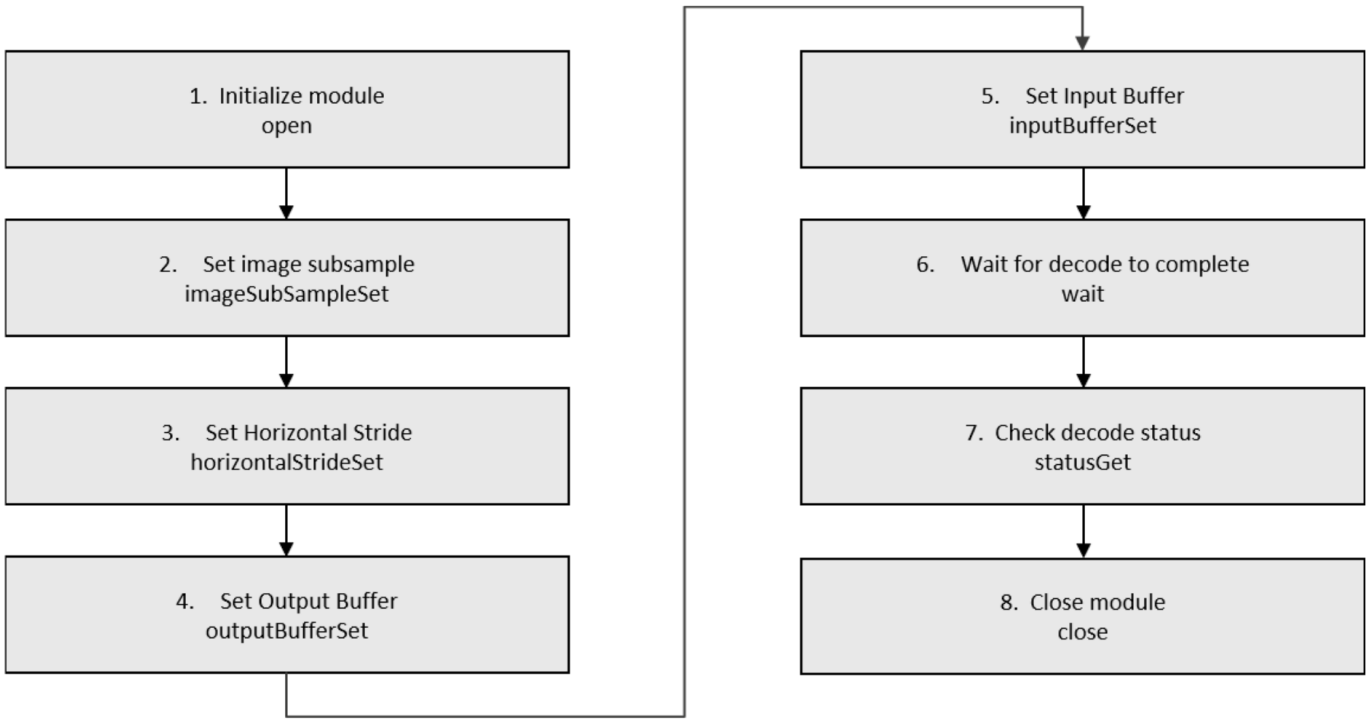


Figure 182: Flow Diagram of a Typical JPEG Decode Framework Module Application

4.1.21 Memory Framework on sf_memory_qspi_nor

4.1.21.1 Memory Framework Module Introduction

The Memory Framework on sf_memory_qspi_nor module provides a high-level API for interfacing with QSPI NOR memory devices. It provides API functions for reading, writing, and erasing data in QSPI NOR Flash memory. The Memory Framework on sf_memory_qspi_nor module is also used by the higher-level Port LevelX Framework on sf_el_lx_nor framework module when support for LevelX wear leveling is required.

Memory Framework Module Features

- Supports memory interface for QSPI NOR flash memory device.
- Supports I/O Operations on QSPI NOR flash memory device.
  - Read
  - Write
  - Erase

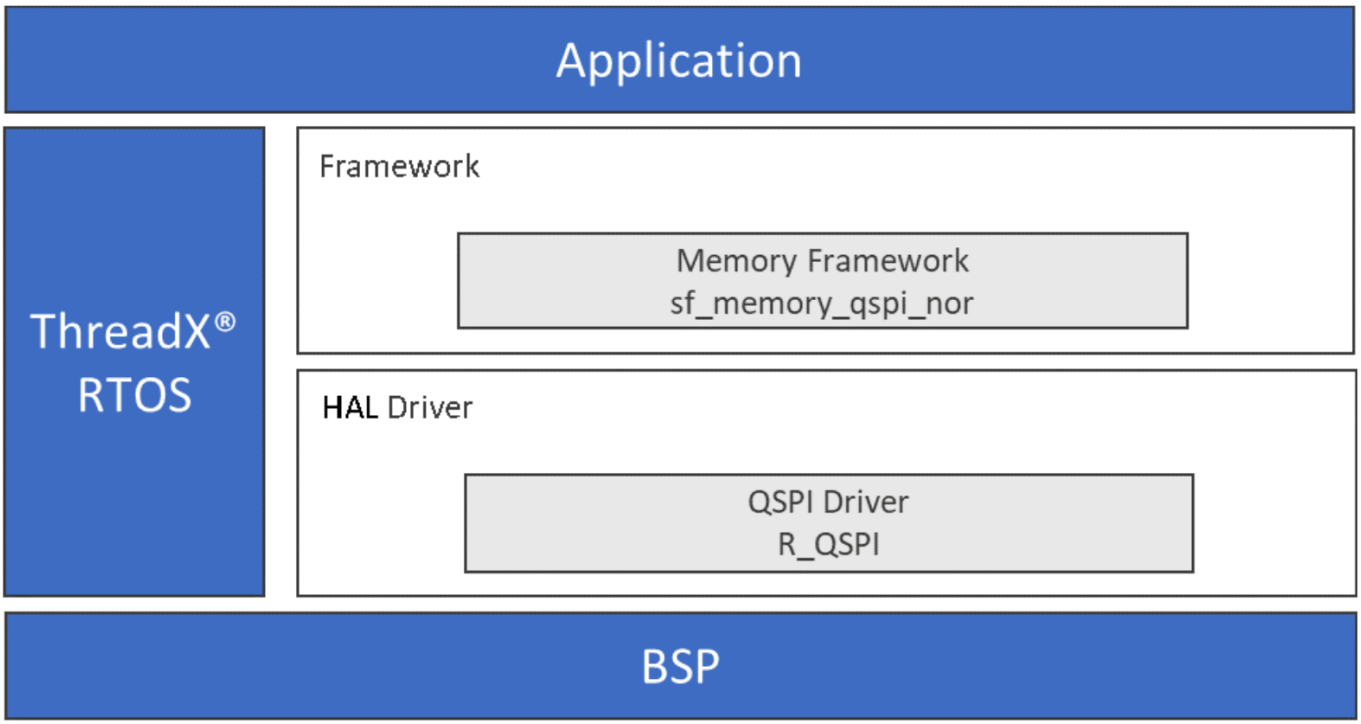


Figure 183: Memory Framework Module Block Diagram

4.1.21.2 Memory Framework Module APIs Overview

The Memory Framework module defines API functions to read, write and erase QSPI NOR flash memory devices. A complete list of the available API functions, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Memory Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_memory_qspi_nor0.p_api->open(g_sf_memory_qspi_nor0.p_ctrl, g_sf_memory_qspi_nor0.p_cfg); Open the SF Memory QSPI NOR driver module for the purposes of reading and writing flash memory.
close	g_sf_memory_qspi_nor0.p_api->close(g_sf_memory_qspi_nor0.p_ctrl); Close the Memory QSPI NOR driver module.
read	g_sf_memory_qspi_nor0.p_api->read(g_sf_memory_qspi_nor0.p_ctrl, p_dest_address, memory_address, num_bytes); Read specified number of bytes of data from a particular address on the QSPI flash device.

write	g_sf_memory_qspi_nor0.p_api->write(g_sf_memory_qspi_nor0.p_ctrl, p_src_address, memory_address, num_bytes); Program data to the flash.
flush	g_sf_memory_qspi_nor0.p_api->flush(g_sf_memory_qspi_nor0.p_ctrl); Flush any pending data to the disk. This is not required for QSPI NOR flash.
erase	g_sf_memory_qspi_nor0.p_api->erase(g_sf_memory_qspi_nor0.p_ctrl, memory_address, num_bytes); Erase a number of bytes from the flash.
infoGet	g_sf_memory_qspi_nor0.p_api->infoGet(g_sf_memory_qspi_nor0.p_ctrl, p_info); Returns the information about the flash.
versionGet	g_sf_memory_qspi_nor0.p_api->versionGet(&version); Get the driver version based on compile time macros.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_ALREADY_OPEN	Driver is already open.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_INVALID_ARGUMENT	Number of bytes requested is invalid.
SSP_ERR_TIMEOUT	Wait timed out.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.21.3 Memory Framework Module Operational Overview

The Memory Framework module supports memory data transfers for a QSPI memory. High-level API functions are available to open, read, write, erase and close the module. If LevelX support is needed, the higher-level Port LevelX Framework on sf_el_lx_nor framework module can be used in conjunction with this module.

The Memory Framework module uses a standard interface that is common for other SSP media

modules. For example, the modules that support SDMMC, SPI Flash and SDRAM/RAM memories use the same API calls, so the programming interface remains the same for any media driver. These modules can be interchanged with one another easily. Device adaptation drivers, such as r_qspi, are accessed through the Memory Framework Interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through memory interface function calls are generally device specific as well.

A user-defined delay callback function can be specified to fine tune the amount of time to wait before starting to poll the QSPI chip after a write or erase operation. This option is provided since there is a large variation in the delay that could be required for a QSPI operation to complete and it varies from chip to chip.

If the delay callback is not specified, then the QSPI driver sleeps for one tick, before polling the status again.

The number of ticks to wait before a timeout error occurs, on a chip erase or write, can be specified in the "Write or Erase Timeout" field. The default is set to 30,000 ticks.

Memory Framework Module Important Operational Notes and Limitations

Memory Framework Module Operational Notes

The media must be erased before using sf_memory_qspi_nor for I/O operations.

Memory Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.21.4 Including the Memory Framework Module in an Application

This section describes how to include the Memory Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Memory Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Memory Framework is g_sf_memory_qspi_nor0. This name can be changed in the associated Properties window.)

Memory Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_memory_qspi_nor0 Memory Framework on sf_memory_qspi_nor	Threads	**New Stack> Framework> Memory> Memory Framework on sf_memory_qspi_nor**

When the Memory Framework on sf_memory_qspi_nor is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or

common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

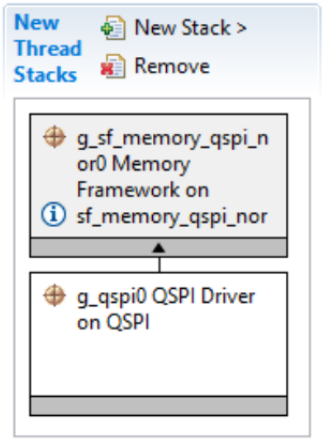


Figure 184: Memory Framework Module Stack

4.1.21.5 Configuring the Memory Framework Module

The Memory Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Memory Framework Module on sf_memory_qspi_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_memory_qspi_nor0	Module name.
Delay Callback (Optional)	NULL	Callback used to add a delay between polling the QSPI chip.
Write or Erase Timeout (in ticks)	30000	Timeout ticks for waiting on write or erase to complete.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Memory Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_qspi0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Memory Framework Module Clock Configuration

The Memory Framework module uses the QSPI peripheral, which uses PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Memory Framework Module Pin Configuration

To use the Memory Framework module, the port pins for the QSPI peripheral must be set as needed. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the Memory Framework Module on sf_memory_qspi_nor

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select **Peripherals**> **Storage:QSPI QSPI0**

4.1.21.6 Using the Memory Framework Module in an Application

The typical steps in using the Memory Framework module in an application are:

1. Initialize the instance using sf_memory_api_t::open API function.

2. Write data to QSPI flash using sf_memory_api_t::write API function.

3. Read data from QSPI flash using sf_memory_api_t::read API function.

4. Erase data from QSPI flash using sf_memory_api_t::erase API function.

5. Read QSPI flash information using sf_memory_api_t::infoGet API function.

6. Close the instance using sf_memory_api_t::close API function.

These common steps are illustrated in a typical operational flow diagram in the following figure:

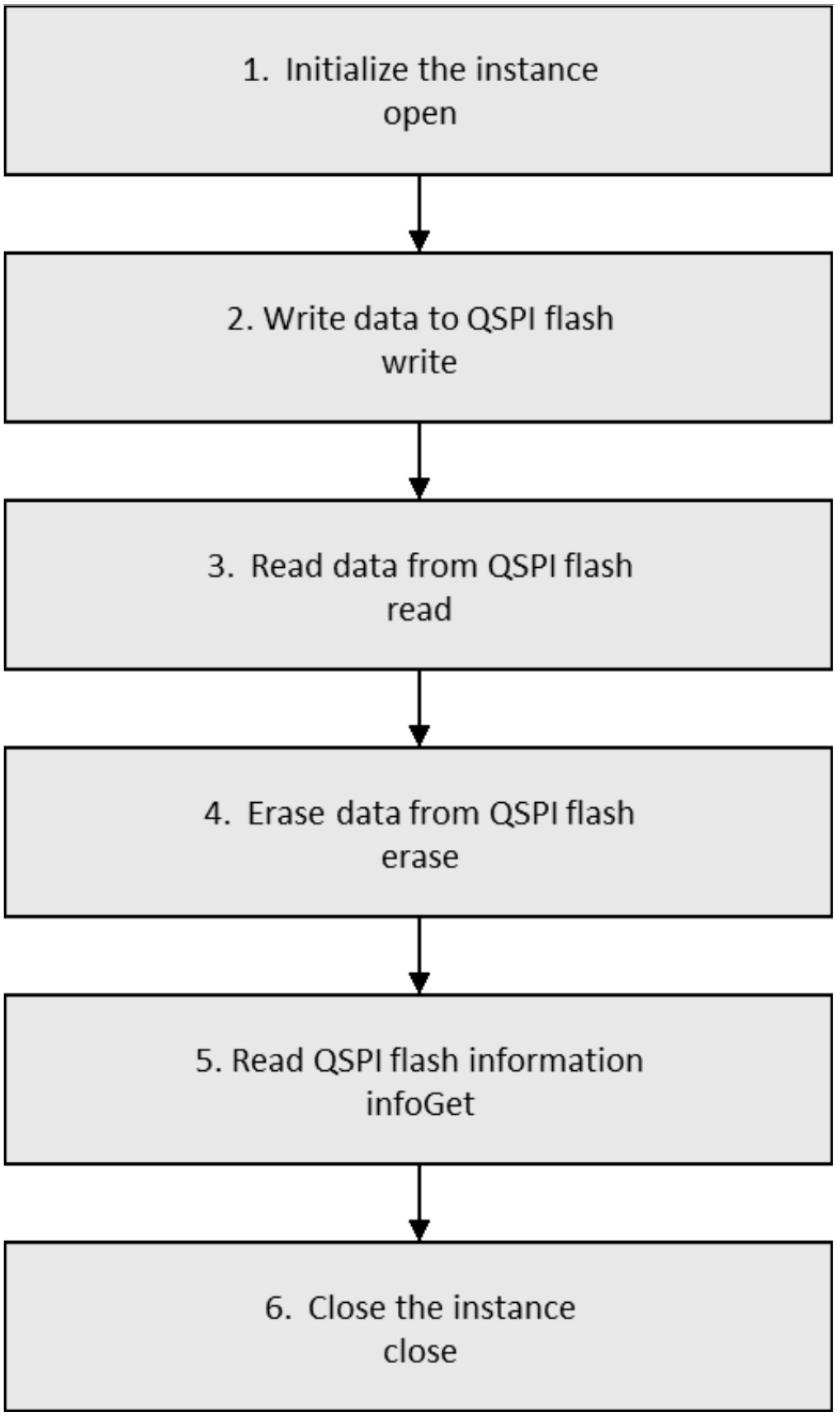


Figure 185: Flow Diagram of a Typical Memory Framework Module Application

4.1.22 Messaging Framework

4.1.22.1 Messaging Framework Module Introduction

The Messaging Framework implements a lightweight and event-driven API for passing messages between threads. The Messaging Framework module allows applications to communicate messages between two or more threads. The framework uses the ThreadX message-queue primitive for message passing and provides more benefits than the ThreadX RTOS message-queue services alone. The Messaging Framework API is purely a software API and does not access any hardware peripherals. The Messaging Framework callback is used to allow an event-producer thread and a message-subscriber thread to handshake after the message passing is done.

The **Messaging** tab is used to either create custom event classes, events and subscribers for the Messaging Framework module or to customize preconfigured events such as the touch event used by the Touch Panel Framework module.

Messaging Framework Module Features

The Messaging Framework module supports the following functions:

- Inter-Thread communication - The framework allows application threads which control disparate devices or manage subsystems to communicate with each other.
- Publishing/Subscribe scheme - The framework design is based on the loosely-coupled messaging paradigm. The design allows multiple threads to listen to an event class. The message producer thread does not need to know who is subscribing to a message for the event class. Subscribers do not need to know who produces the message.
- Message management - The framework supports buffer control blocks to manage each message including flags to control the buffer and a callback function pointer for handshaking.
- Message buffering - The framework manages buffer allocation and release for messaging. An application can make use of the allocated buffer to write a message and discard the message if it is no longer needed.
- Synchronous communication - The framework supports asynchronous messaging by using the ThreadX message-queue but also provides an option to create a handshake between a message producer and a subscriber thread. The handshake is implemented by invoking a user-callback function of the producer thread from a subscriber thread.
- Message formatting - The framework provides a predefined common message header. It also provides some typical payload structure templates as examples.
- Message Priority - The framework can send a high-priority message so that a subscriber thread can retrieve the message prior to other messages which are located in the message queue.

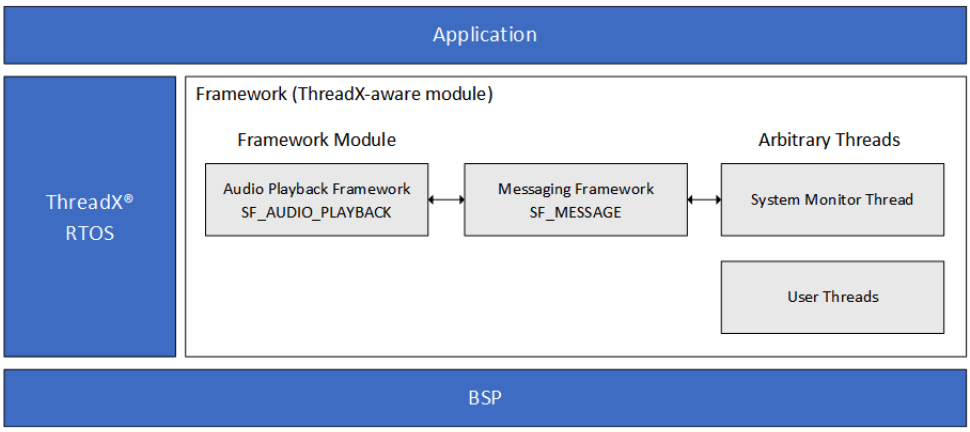


Figure 186: Messaging Framework Module Block Diagram

4.1.22.2 Messaging Framework Module APIs Overview

The Messaging Framework module defines APIs for opening and closing the framework, acquiring and releasing buffers, and posting messages to subscribers. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Messaging Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_message.p_api->open (g_sf_message.p_ctrl, g_sf_message.p_cfg); Initialize message framework. Initiate the messaging framework control block, configure the work memory corresponding to the configuration parameters.
close	g_sf_message.p_api->close (g_sf_message.p_ctrl); Finalize message framework.
bufferAcquire	g_sf_message.p_api->bufferAcquire (g_sf_message.p_ctrl, &p_buffer, &acquire_cfg, wait_option); Acquire buffer for message passing from the block.
bufferRelease	g_sf_message.p_api->bufferRelease (g_sf_message.p_ctrl, &p_buffer, option); Release buffer obtained from SF_MESSAGE_BufferAcquire.
post	g_sf_message.p_api->post (g_sf_message.p_ctrl, (sf_message_header_t *) p_payload, &post_cfg, &err_post, wait_option); Post message to the subscribers.

pend	g_sf_message.p_api->pend (g_sf_message.p_ctrl, &my_queue, (sf_message_header_t **)&p_buffer, &p_header, wait_option); Pend message.
versionGet	g_sf_message.p_api->versionGet (&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	Required pointer is NULL.
SSP_ERR_BUFFER_RELEASED	The buffer is released.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.
SSP_ERR_IN_USE	The messaging framework is in use.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
SSP_ERR_INVALID_WORKBUFFER_SIZE	Invalid work buffer size.
SSP_ERR_MESSAGE_QUEUE_EMPTY	Queue is empty. (Timeout occurs before receiving a message if timeout option is specified.)
SSP_ERR_MESSAGE_QUEUE_FULL	Queue is full. (Timeout occurs before sending a message if timeout option is specified.)
SSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
SSP_ERR_NO_SUBSCRIBER_FOUND	No subscriber found.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_TOO_MANY_BUFFERS	Too many message buffers.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.22.3 Messaging Framework Module Operational Overview

The following figure shows the overview of the messaging data flow between a message producer thread and subscriber thread(s) in the system making use of the Messaging Framework module.

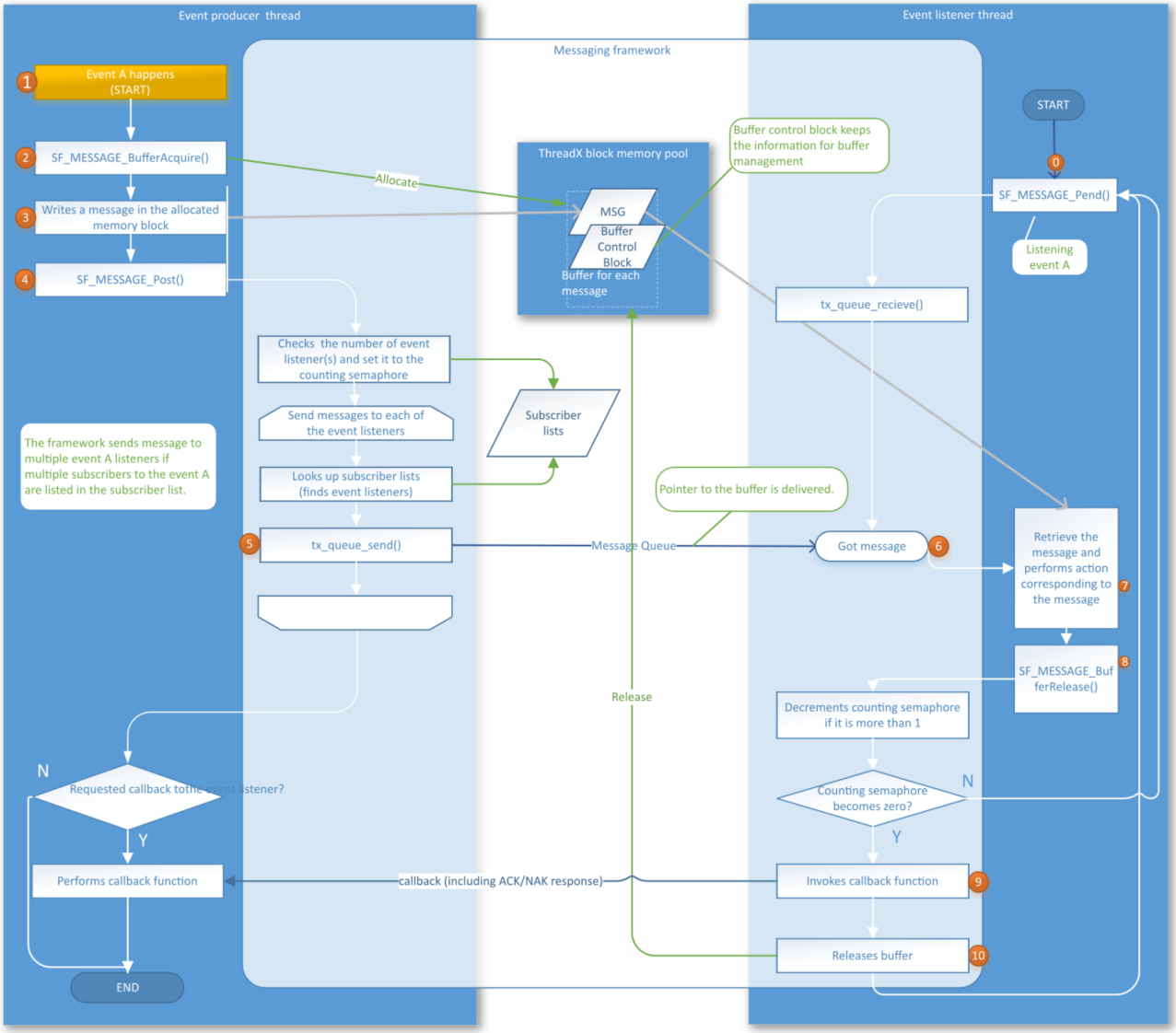


Figure 187: Messaging Framework Module Data Flow

The following is a description for each stage of the message passing procedure:

Note: A thread in the system has been called using the sf_message_api_t::open API and message subscriber threads have called the sf_message_api_t::pend API to pend on a message for the event class.

1. An event (Event A) happens on a message producer thread.
2. A message producer thread calls sf_message_api_t::bufferAcquire to acquire a buffer from the ThreadX memory pool managed by the Messaging Framework module; sf_message_api_t::bufferAcquire returns the address of allocated buffer.
3. A message producer writes the message to the allocated buffer.

4. A message producer calls sf_message_api_t::post to post the message.
5. The Messaging Framework module looks up the event subscriber list and sends a message to the message queue of the message subscriber threads using the ThreadX message-queue primitive. The framework just sends the pointer to the buffer but does not send the entire message, thereby performing lightweight message passing.
6. The message reaches the message queue of the message subscriber threads and the message subscriber threads return from sf_message_api_t::pend. The API function returns the buffer address where the message is stored to message subscriber threads.
7. The message subscriber threads receive the message and perform an action corresponding to the event.
8. The message subscriber threads call sf_message_api_t::bufferRelease to try to release the allocated buffer for the message. If the message subscriber thread is not the last one subscribing to the message, the framework does not release the buffer as the message has to be kept in the buffer until all subscribers have received the message.
9. The Messaging Framework module invokes a user-callback function which is specified by an event producer thread if the message subscriber thread is the last one in the message subscriber group.
10. The Messaging Framework module releases the buffer if all the subscribers in the group have notified the framework that they have consumed the message. (Note that the release option SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE in the sf_message_api_t::bufferRelease API function should not be used in a multiple subscriber scenario.)

Messaging Framework Module Message Producer and Subscribers

The Message Framework module is an inter-thread messaging system based on the publish/subscribe model. A message is posted with an event class code by an event producer thread. The message subscriber threads can check for pending messages which subscribe to the event class. Subscribers are registered in the subscriber list, which is referred to by the framework. The subscriber list allows the framework to deliver a message to multiple subscribers.

Every thread which joins the Messaging Framework module system network can send a message, and all threads in the network can listen to the message.

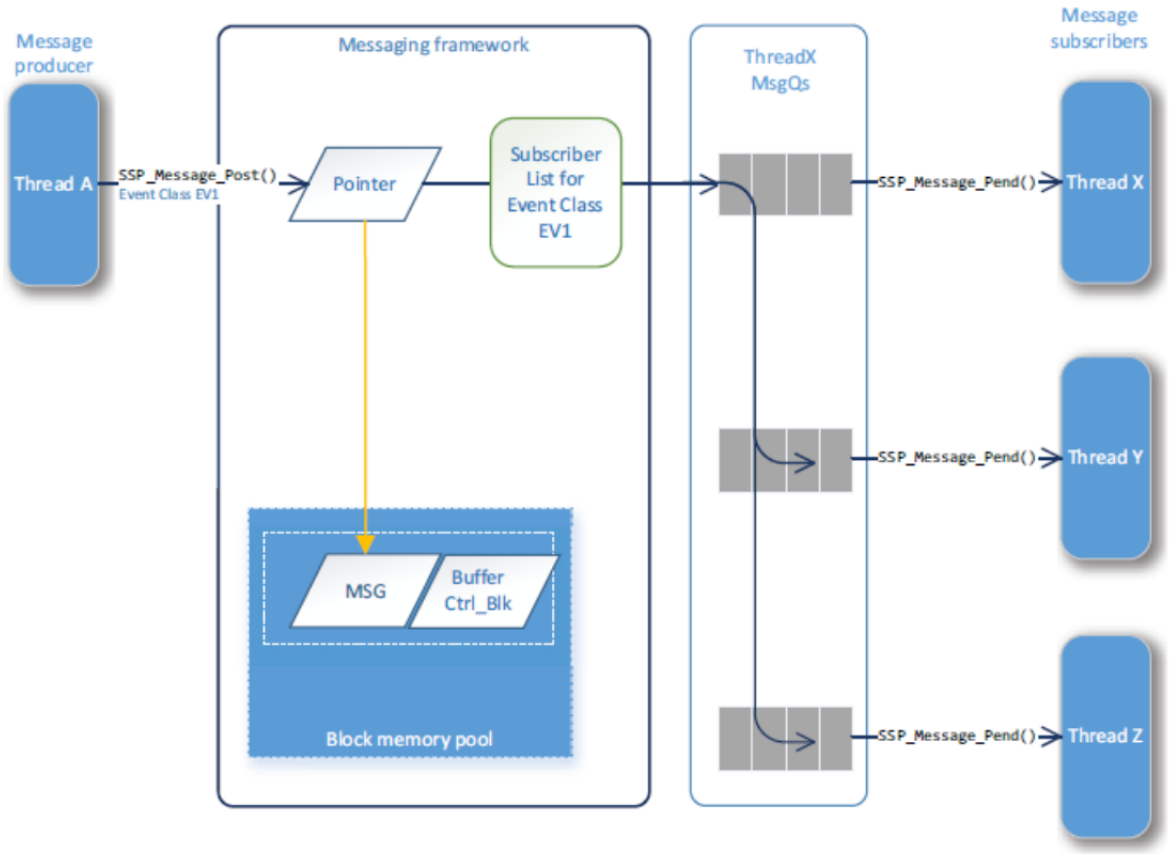


Figure 188: Messaging Framework Module Subscribing

Messaging Framework Module Events, Subscribers, and Messages

Messaging Framework Module Event Class Code

The event class code is the most important definition for the Messaging module. The Messaging module uses the event class code as the mechanism to connect a message producer with subscribers; the event class code is the class definition of the events which occur in the application. The classification of the event class relies on the user definition, but it is intended to be the group name of the particular events which can occur in a subsystem.

For example, you can use the 'Audio Playback' event class, which is part of the Audio Playback Framework module. The 'Audio Playback' Event Class is automatically loaded into the event classes window. This window is available on the Messaging tab in the Project Configurator when you add the Audio Playback Framework to your Synergy project.

The event class code is defined in the sf_message_event_class_t enumeration and has a prefix SF_MESSAGE_EVENT_CLASS_XXX. Since the definition of the event class code is different for each system, the framework does not provide a concrete event class code but instead provides a set of event class codes as examples. (See the Configuring the Messaging Framework Module section.) The maximum number for the event class is 255.

An application can use the event class code as follows:

- The message producer thread sets the event class code to the event_b.class_code bit fields in the sf_message_header_t type common message header before posting the message.
- The message subscriber-thread branches to the event processing corresponding to the event class code which is set to the message header after receiving the message.
- The subscribers for the event class code must be grouped and registered in the subscriber list so that the Message Framework can deliver the message to the subscribers.

The following figure shows how you can configure an event class using the ISDE:

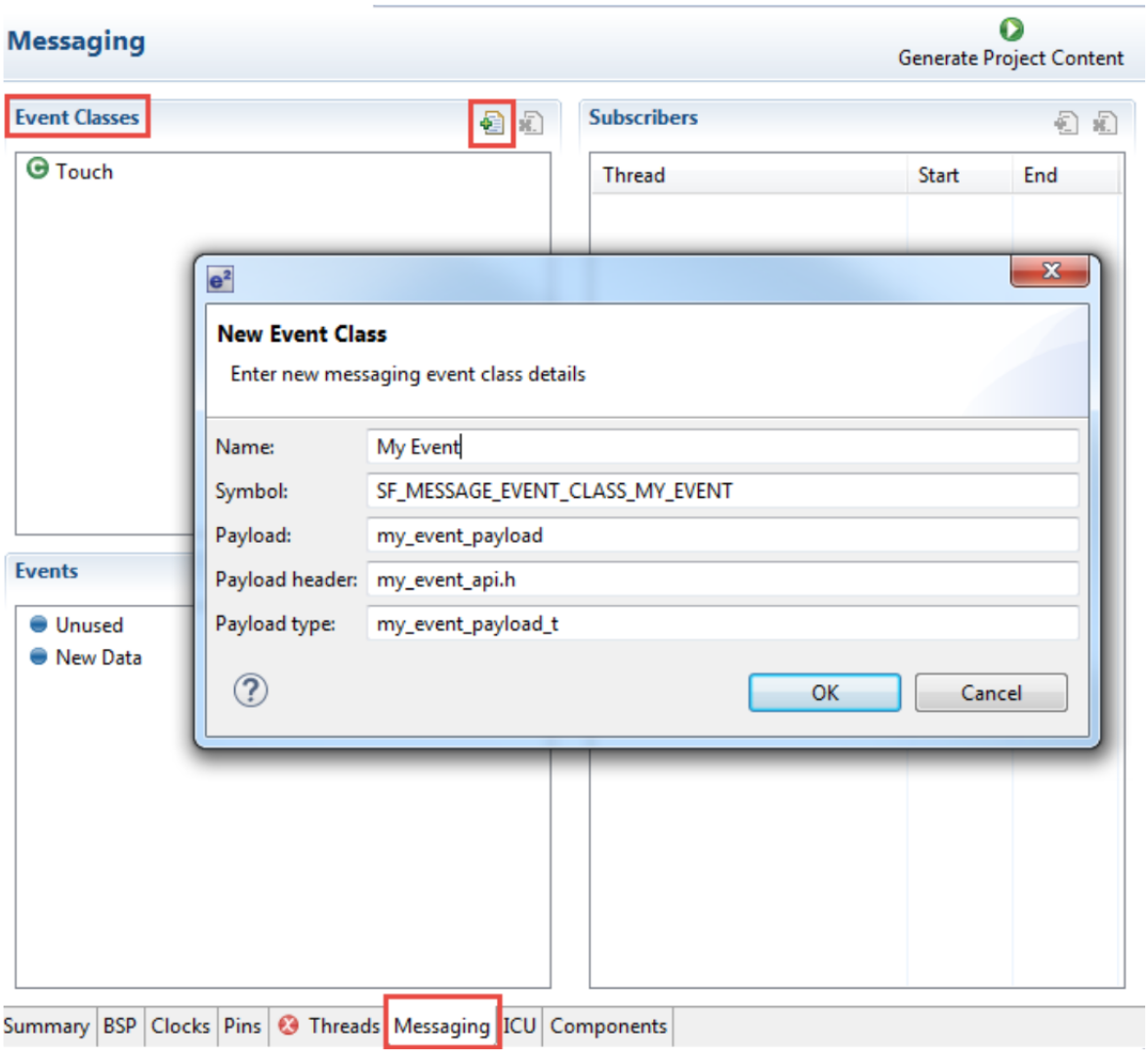


Figure 189: Messaging Framework Module ISDE Event Class Configuration

Messaging Framework Module Event Class Instance Number

The event class instance number is used when an application needs to have different event class instances. For example, the audio streaming event class can have instance *N* which represents the streaming channel N. Message subscribers can receive a message only if the event class instance number in the message common header matches to the number it owns.

In other words, messages for which the event class instance number is out of range for the subscriber are filtered out and not delivered to the subscriber even though it is the event class subscriber. The maximum for the event class instance number is 255.

Note

The Touch Panel Framework typically only has one instance, and therefore the event class instance number is 0 with the start and end values of the subscriber list set to 0.

An application can use the event class instance number as follows:

- The message producer thread sets the event class instance number to the event_b.class_instance bit fields in the sf_message_header_t type common message header before posting the message.
- Each subscriber instance in the Subscriber List has to specify the range of the event class instance numbers to receive the message (sf_message_subscriber_t::instance_range.start and sf_message_subscriber_t::instance_range.end).
- If there is no need for multiple instances for an event class, just specify zero to sf_message_header_t::event_b.class_instance, sf_message_subscriber_t::instance_range.start, and sf_message_subscriber_t::instance_range.end in the subscriber instance for the subscriber list.

Messaging Framework Module Event Code

The event code includes the details of the event definition. For instance, the event codes for the audio playback event class are "playback start" and "playback stop." Another example is 'set' or 'get' for the 'time' Event Class. The event code is enumerated in the sf_message_event_t and has a prefix SF_MESSAGE_EVENT_XXX. The definition of the event code relies on the user code as well as the event class code. The framework provides some code as examples. See Configuring the event class code and event code for configuring the event code. The maximum for the event class instance number is 65535.

An application can use the event code as follows:

The message producer thread sets the event code to the event_b.code bit fields in the sf_message_header_t type common message header before posting the message.

The message subscriber thread performs an action corresponding to the event code which is set to the message header after receiving the message.

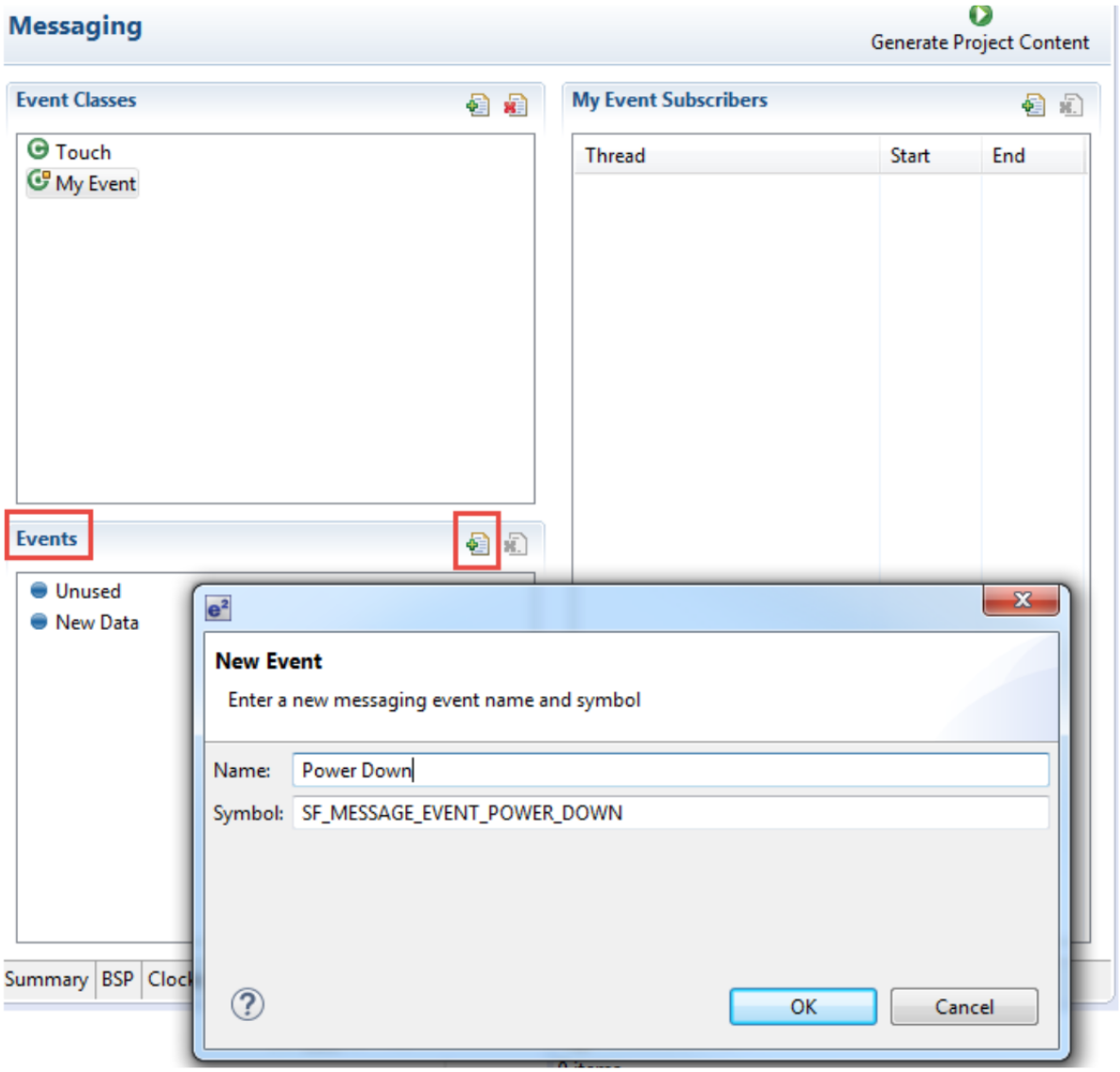


Figure 190: Messaging Framework Module ISDE Event Configuration

Messaging Framework Module Subscriber List

The subscriber list is used for message delivery and is looked up by the framework.

The framework starts to look up the message queues of each subscriber thread from the subscriber group listed in the head of the pointer array to the sf_message_subscriber_list_t instance. The important point of the subscriber list is that it is grouped by event class code (event_class).

When the framework looks up the subscriber list in the post API function at runtime, it compares the Event Class code in the message header (sf_message_header_t::event_b.class), which is included in the message payload data, with the one in the subscriber group instance (event_class). If there is a match, the Framework goes to the next level to get the message queue instance (sf_message_subscriber_t::queue) until the iterations reach number_of_nodes). If there is no match, the framework looks up the next subscriber group and continues until encountering a NULL in the pointer array to the sf_message_subscriber_list_t instance.

In the look-up procedure, the subscriber group listed at the head of the Subscriber List gets the

highest throughput for messaging, but lower subscriber groups encounter a penalty and get lower throughput for messaging.

The subscriber list is the look-up table for all message subscribers. The subscriber list is configured at compile time. It is statically mapped to the memory and looked up by the framework when the post API function is called. The subscriber list allows the framework to determine message queues to deliver a message to. The subscriber list consists of two structures sf_message_subscriber_list_t and sf_message_subscriber_t as shown in the following figure:

- A queue for a subscriber thread is registered in sf_message_subscriber_t instance.
- The instances above for the same event class code are grouped and listed in a pointer array.
- The pointer array for a subscriber group is registered in a sf_message_subscriber_list_t instance.
- The subscriber list is the pointer array to the subscriber group structures. Subscribers are grouped by the event class code.
- The pointer array must be terminated by NULL.

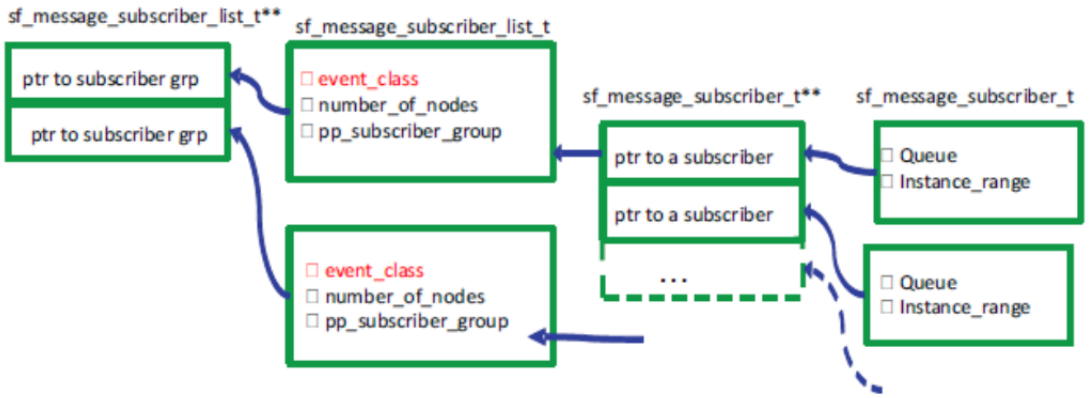


Figure 191: Messaging Framework Module Subscriber List

In the ISDE, you can configure a subscriber grouped by the event class for the thread you named in the **Threads** tab. In the following example, the thread is named "My Thread" in the **Threads** tab. The start and end values reflect the event class instance numbers this thread accepts.

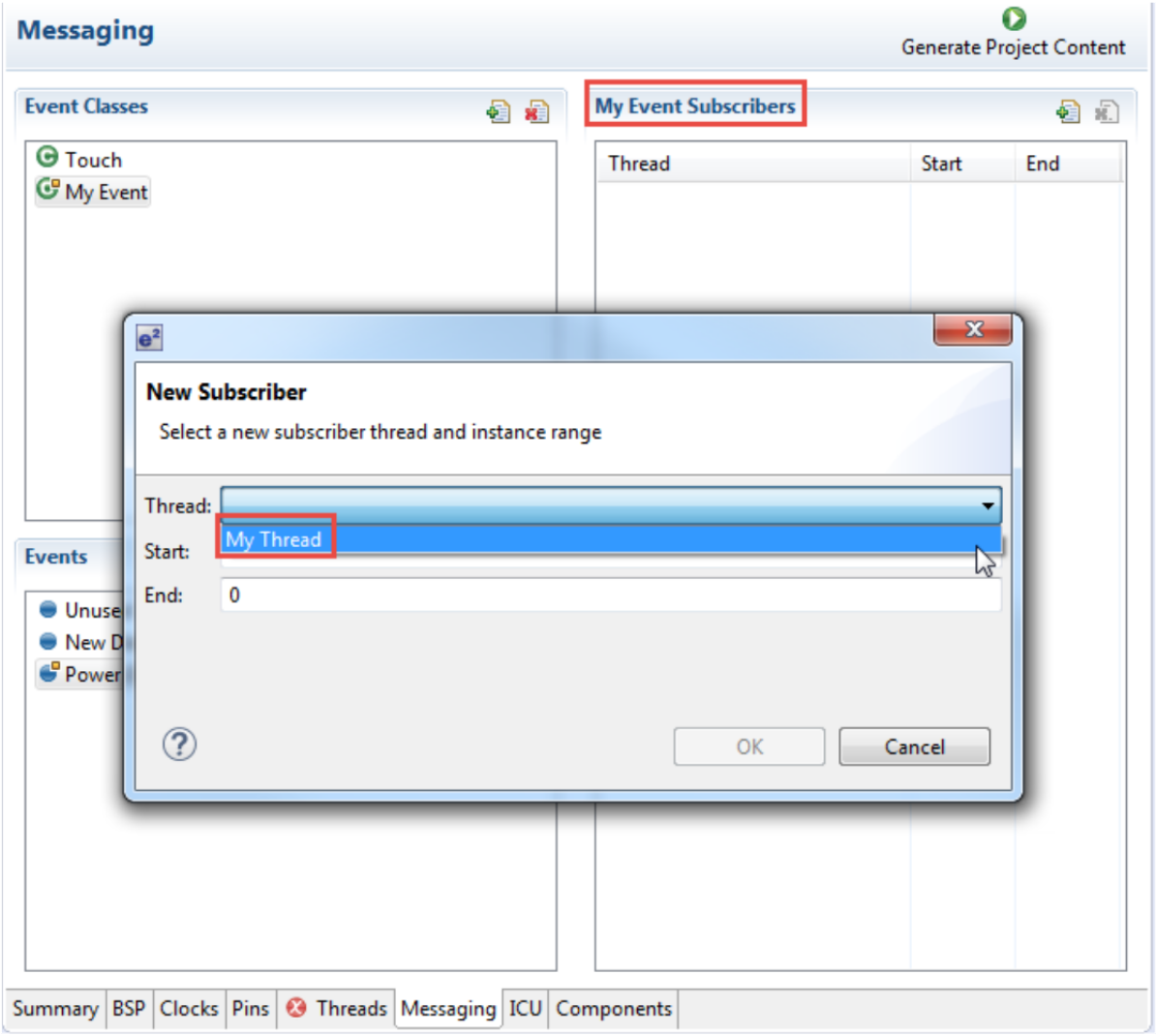


Figure 192: Messaging Framework Module ISDE Subscriber Configuration

Messaging Framework Module Message Payload

The message payload is structured data used by the message producer and the message subscribers to communicate with each other. The message payload contains event class and event code in the common header (sf_message_header_ttype data, see event class and event code) so that a message producer can post a message to the subscribers to inform the subscribers which event happened.

You must define a system specific message payload structure except for modules for which the SSP provides predefined structures such as the Audio Playback Framework module. The message payload can contain additional data, which are required for the event processing, in addition to the common header.

Messaging Framework Module SSP Predefined Payload

The SSP contains the predefined message payload structure sf_audio_playback_data_t type for the Audio Playback Framework module.

The Audio Playback Framework module uses the Messaging Framework internally and defines a

suitable message payload structures. An application thread that posts audio event messages to the Audio Playback Framework module can use sf_audio_playback_api.h.

Messaging Framework Module User-Defined Payload

You must define a message payload structure for each event class code; exceptions include the SSP predefined payloads described previously or payloads that require only the common header. To create a new message payload structure, add a common message header (sf_message_header_t type structure) at the head in the user-specific message payload structure. The size of the header is 4 bytes.

Note

The payload size must not be greater than the buffer size.

The buffer size limit is critical; oversized data written beyond the buffer may destroy data in the block memory pool, which is required by ThreadX kernel. Violating the size limit results in a hard fault exception. The buffer size can be configured insf_message_ctrl_t::buffer_size.

Messaging Framework Module Important Operational Notes and Limitations

Messaging Framework Module Operational Notes

Messaging Framework and OS Message Queue Service

The Messaging Framework module uses the ThreadX primitive-message queue and kernel services and supports some enhancement over the ThreadX RTOS features. For this reason, the Messaging Framework module does not work exactly the same as the ThreadX message-queue service. However, a messaging system with the Messaging Framework module can work simultaneously with the ThreadX message queue services in an application if the two messaging systems are separated.

API Calls Contexts

- The sf_message_api_t::open API can only be called from a thread; it can be called only once per the message framework control block instance. The behavior is undefined if the function is called twice.
- The sf_message_api_t::close API can only be called from a thread.
- The sf_message_api_t::bufferAcquire API can be called from a thread and an ISR.
- The sf_message_api_t::bufferRelease API can only be called from a thread.
- The sf_message_api_t::post API can be called from a thread and an ISR.
- The sf_message_api_t::pend API can be called from a thread and an ISR.

Estimating the Number of Buffers

The number of buffers available to be allocated in the work memory should be estimated properly when designing the messaging system. The number of buffers is estimated as follows:

$$N \approx \frac{W_m}{M_h + B_{ch} + T_v} = \frac{W_m}{M_h + 12\ bytes + 4\ bytes}$$

where:

- N – number of buffers,
- W_m – work memory size (in bytes),
- M_b – message buffer size (in bytes),
- Bcb = 12 bytes – size of buffer control block,
- T_x - 4 bytes – reserved bytes for ThreadX.

The maximum number possible for buffers allocated at the same time equals the total amount of depth of message queues in the system. Ideally, the number of buffers for a robust system should be the sum of the depths of the message queues in the system.

Message Queue Size and Depth Setting

The Messaging Framework module needs a 4-byte memory block on the message queue as it delivers the pointer to the buffer which contains a message payload. For this reason, the size of the message queue is fixed to 4 bytes.

The depth of the message queue is arbitrary, but it should accommodate the number of queued messages at runtime. As a guideline, estimate the value as follows:

$$D \approx \frac{P_{avg}}{S_{avg}}$$

where:

- D – queue depth,
- P_{avg} – average message delivery rate from producers,
- S_{avg} – average event loop completion time in the subscriber.

Messaging Framework Module Limitations

- In a multi-threaded and repeated message transfer application, keep the message Producer thread priority higher than the subscriber thread. So that the producer thread completes all its operation before the subscriber thread starts releasing the received buffer and avoid sending the message to the wrong queue.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.22.4 Including the Messaging Framework Module in an Application

This section describes how to include the Messaging Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Messaging Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Messaging Framework is g_sf_message0. This name can be changed in the associated Properties window.)

Messaging Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_message0 Messaging Framework on sf_message	Threads	**New Stack> Framework> Services> Messaging Framework on sf_message**

When the Messaging Framework on sf_message is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

Note: While using sf_message framework for communication between the multiple threads having different priorities,the priority of the thread in which sf_message stack is added should be kept high.

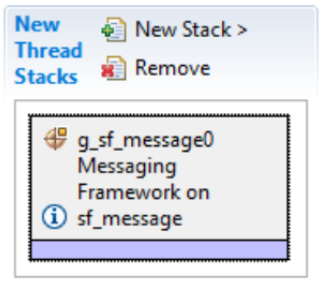


Figure 193: Messaging Framework Module Stack

4.1.22.5 Configuring the Messaging Framework Module

The Messaging Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Messaging Framework Module on sf_message

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Message Queue Depth (Total number of messages to be enqueued in a Message Queue)	16	Specify the size of Thread X Message Queue in bytes for Message Subscribers. This value is applied to all the Message Queues.
Name	g_sf_message0	The name of Messaging Framework module control block instance.
Work memory size in bytes	2048	Specify the work memory size in bytes. Choosing a small number results in a small number of buffers which can be allocated at the same time (You can confirm the total buffer number on: sf_message_ctrl_t:: number_of_buffers). If the value is smaller than the peak number of messages to be posted at the same time, the Framework has a buffer allocation failure affecting system performance.
Pointer to subscriber list array	p_subscriber_lists	Specify the name of pointer to the Subscriber List array.
name of the block pool internally used in the messaging framework	sf_msg_blk_pool	The name of the memory block memory the Framework creates in the control block. This parameter might be useful for debugging purpose but NULL can be specified for saving memory.
Name of generated initialization function	sf_message_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Messaging Framework Module Creating a Messaging Queue

The messaging configurator automatically creates the message queue for the subscribers.

Messaging Framework Module Configuring an Event Class and Event

To use the Messaging Framework with your own event class, use the **Threads** tab and the **Messaging** tab of the project configurator in the ISDE.

In the **Threads** tab, do the following:

1. Add the Messaging Framework component in the Thread Stacks panel of the **Threads** window.
2. Add a new thread in the **Threads** window and give it a unique name.
3. In the **Messaging** tab (see event class code), do the following:
   a. In the **Event Class** window, add an event.

   b. Enter the name of the event class for your thread to subscribe to in the **New Event Class** dialog box.

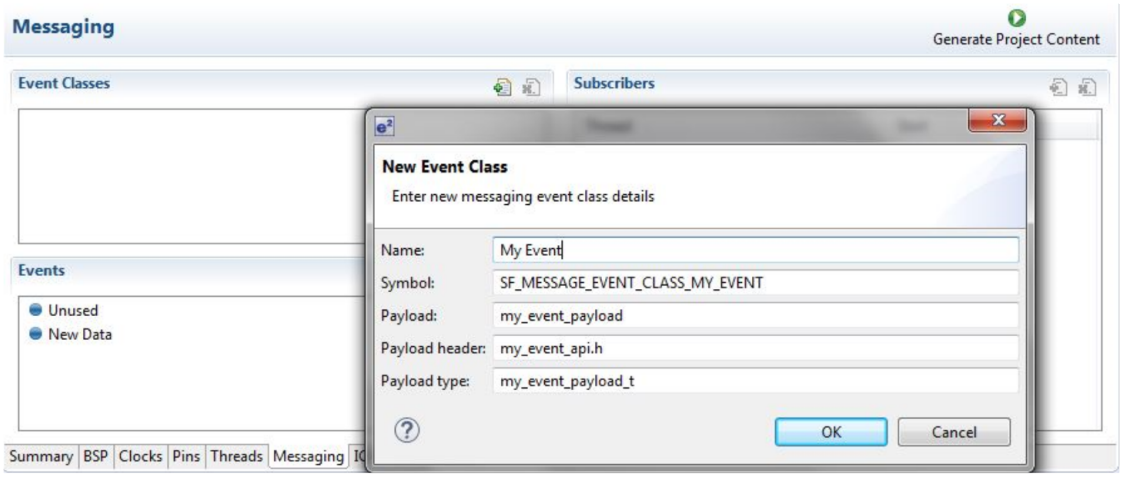


Figure 194: Messaging Framework New Event Class Configuration

4. In the **Events** window, add any events that your application may support (see event code).

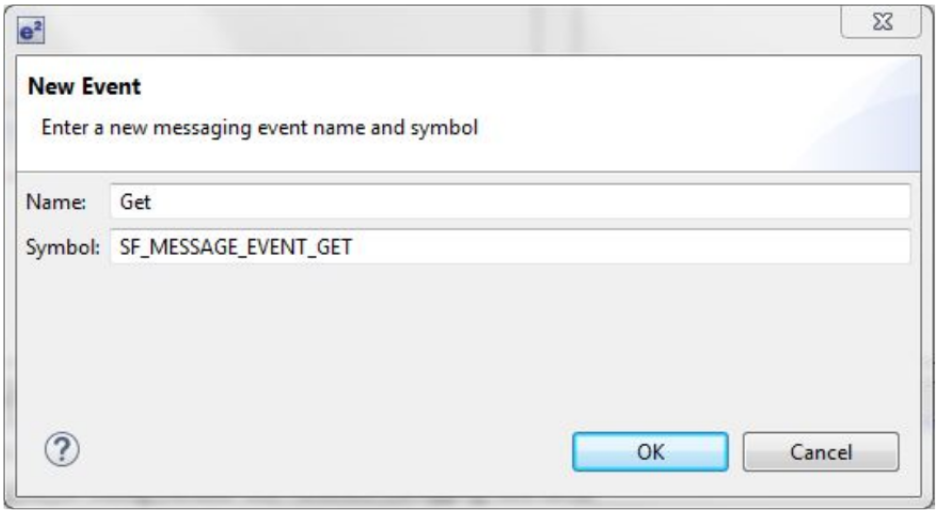


Figure 195: Messaging Framework New Event Configuration

Your custom event class code and event code are stored in a file named sf_message_port.h. The audio playback and the touch event classes are two predefined event classes in the SSP. The Touch Event Class only uses the new data event, SF_MESSAGE_EVENT_NEW_DATA.

Messaging Framework Module Configuring the Subscriber List

In the **Messaging** tab (see also Subscriber List), do the following:

1. Select the event class in the **Event Classes** window and configure a thread for the subscriber list in the event class **Subscribers** window.
2. Select your thread from the drop-down list in the **Threads** dialog box.
3. Next to Start, enter the start number of the event class instance(s). If your system does not use multiple event class instances for the event class, or you are not sure what number to specify, just keep the default number (0.) Allowed values range from 0 to 255.
4. Next to End, enter the last number of the event class instance(s). If your system does not use multiple event class instances for the event class, or you are not sure what number to specify, just keep the default number (0.) Allowed values range from 0 to 255.
5. Click **OK**. A subscriber for your specified event is added in the subscriber list.
6. Repeat these steps for all event class instances if there are more than one.

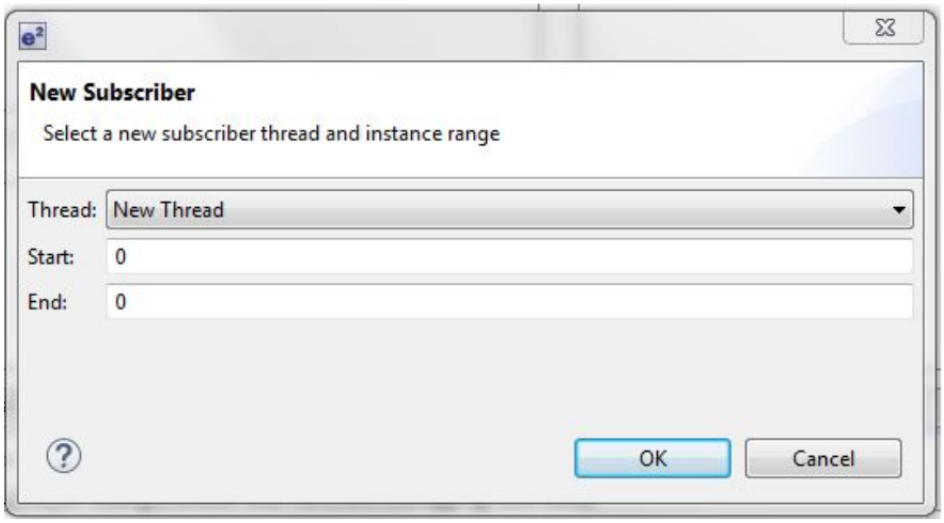


Figure 196: Messaging Framework New Subscriber Configuration

Messaging Framework Module Configuring the Event Class Code and Event Code

Messaging Framework Module Defining the Message Payload

You can define your own message payload structure. Every user-defined message structure must include the sf_message_header_t type structure as one of the members, but the other members are entirely user-definable. The Messaging Framework does not care where the message payload structures are defined. You can include the file which defines your own message payload structure in the source file for your message producer and subscriber threads.

Opening the Messaging Module in the Messaging Framework Module

Configure the sf_message_cfg_t type configuration parameters to match your system. You can generate code for the configuration structure through the Synergy Configuration tool. Add a Messaging Framework component to the thread stacks in the **Threads** tab and modify the properties for the Messaging Framework module in the Properties window. When you press the **Generate**

Project Content button, the code for the Messaging Framework module is generated on the thread code.

Messaging Framework Module Acquiring a Buffer

Before posting a message, an event producer thread must acquire a buffer for the message from the Messaging Framework module. An event producer thread can acquire the buffer by calling sf_message_api_t::bufferAcquire.

When the API function returns SSP_SUCCESS, the buffer with message buffer size in bytes configured on Synergy Configuration tool is allocated in the memory pool managed by the Messaging Framework. The maximum number allowed to be allocated depends on the configuration Work memory size in bytes specified on the Synergy Configuration tool. For the estimation of the maximum number, see *Estimating the Number of Buffers*.

The sf_message_api_t::bufferAcquire API has several options to change the message passing behavior:

- **buffer keep:** This option allows the application thread to hold the buffer not to be released by the API function sf_message_api_t::bufferRelease if set to true. Typically, the buffer is to be released by sf_message_api_t::bufferRelease when the message passing is done; however, in a scenario to have periodical or repeated message passing between threads, we can reuse the same buffer for the messaging without allocating and releasing the buffer each time. Enabling this option reduces the overhead in the buffer allocation/release operation and improve the system throughput.
- **wait_options:** This is the wait time option which is valid if all buffers have been acquired. Any arbitrary thread tick count, TX_WAIT_FOREVER, TX_NO_WAIT can be set for this option. For details, see the tx_block_allocate() description for the ThreadX service call in the *ThreadX User Guide*.

Messaging Framework Module Releasing a Buffer

After message subscriber threads receive a message posted by an event producer, the message subscriber threads must release the buffer to the framework. Buffer releasing is performed by calling sf_message_api_t::bufferRelease. Since the API function can be called multiple times if there are multiple event subscribers in the system, the actual buffer release is performed only by the last message subscriber thread in the event subscribers. For instance, if there were three subscribers in the subscriber group for the event class, the first and second thread which call sf_message_api_t::bufferRelease do not release the buffer. Only the third thread releases the buffer. Note, if the buffer keep option is specified by sf_message_api_t::bufferAcquire, the buffer is never being released except when option SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE is passed to the API function argument option. (Also see Messaging Framework callbacks for sf_message_api_t::bufferRelease API function usage.)

The API is also used for invoking a user-callback function to create a handshake between an event producer thread and a message subscriber thread.

Posting a message

1. After getting a buffer by sf_message_api_t::bufferAcquire, an event producer can write the message payload data to the buffer location.
2. **ATTENTION**: Writing data to the buffer is the user's responsibility and writing more data than the buffer size causes a fatal error in the Messaging Framework module.
3. Write an event class code to the sf_message_header_t::event_b.class_code in the payload structure.

4. Write an event code to the sf_message_header_t::event_b.code in the payload structure. It is not mandatory to specify this but necessary in most cases.
5. Write an event class instance number to sf_message_header_t::event_b.class_instance. Specify a number from 0 to 255 if your system has multiple instances for an event class. Specify 0 if your system simply uses single event class instance.
6. Post the message by the post API. Note that the pointer to the buffer needs to be casted with sf_message_header_t * type when given to the API. The message will be delivered to the message subscribers which are registered in the message subscriber list. The post API has several options to change the message passing behavior.
    - **Message priority:** Message can take two level message priority, SF_MESSAGE_PRIORITY_NORMAL or SF_MESSAGE_PRIORITY_HIGH. When SF_MESSAGE_PRIORITY_HIGH is specified, the message is queued at the front of the message queue of the message subscriber. This is typically used for the emergency message to make the message subscribers handle the event prior to the events which might have been queued in the message queue.
    - **User-callback function:** This function is registered in the buffer control block of the Messaging Framework module. The callback function is invoked by bufferRelease. This function can be used for handshaking between an event producer thread and a message subscriber thread.
    - **Wait_options:** This is the wait time option which is valid if a message queue of the message subscriber thread is full. Any arbitrary ThreadX tick count, TX_WAIT_FOREVER and TX_NO_WAIT can be set to this option. For details, see the description of tx_queue_send() ThreadX service call in ThreadX User Guide.

Checking for a Pending Message

1. After the Messaging Framework module is opened, the message subscriber threads can wait for a message by calling pend. In general use, the second API argument specifies the pointer to a message queue, which you configured for the message subscriber thread in the Thread Subscribers pane in the Messaging tab, but you can specify the other message queues instead if required.
2. When a message is delivered from an event producer, the thread returns from pend.
3. The API returns the pointer to the buffer which contains the message to the thread through the third argument of the API.
4. The message subscriber casts the pointer above with a pointer type for the user custom message payload structure and does the event processing corresponding to the Event Class code sf_message_header_t::event_b.class_code, Event code sf_message_header_t::event_b.code and the user defined arbitrary data in the message.

Note that pend has the wait_option to change the behavior of the API function:

The fourth argument of **pend** is the wait time option, which is only valid if the message queue of the message subscriber thread is empty. Any arbitrary ThreadX tick count, TX_WAIT_FOREVER, and TX_NO_WAIT can be set to this option. For details, see tx_queue_send()ThreadX service call in the ThreadX User Guide.

Messaging Framework Module Interrupts

The Messaging Framework module does not use any interrupts.

4.1.22.6 Using the Messaging Framework Module in an Application

The steps in using the Messaging Framework on sf_message module in a typical application are:

- Create a Message Queue
- Configure the Event Class and Event
- Configure the Subscriber List
- Configure the Event Class Code and Event Code

Once configuration is complete, the module's APIs can be used in the target application as follows:

1. Initialize the Messaging Framework with the sf_message_api_t::open API
2. Acquire a buffer with the sf_message_api_t::bufferAcquire API
3. Post a message with the sf_message_api_t::post API
4. Check for a pending message with the sf_message_api_t::pend API
5. Release a buffer using the sf_message_api_t::bufferRelease API
6. Close the Messaging Framework with the sf_message_api_t::close API

These common steps are illustrated in a typical operational flow diagram in the following figure:

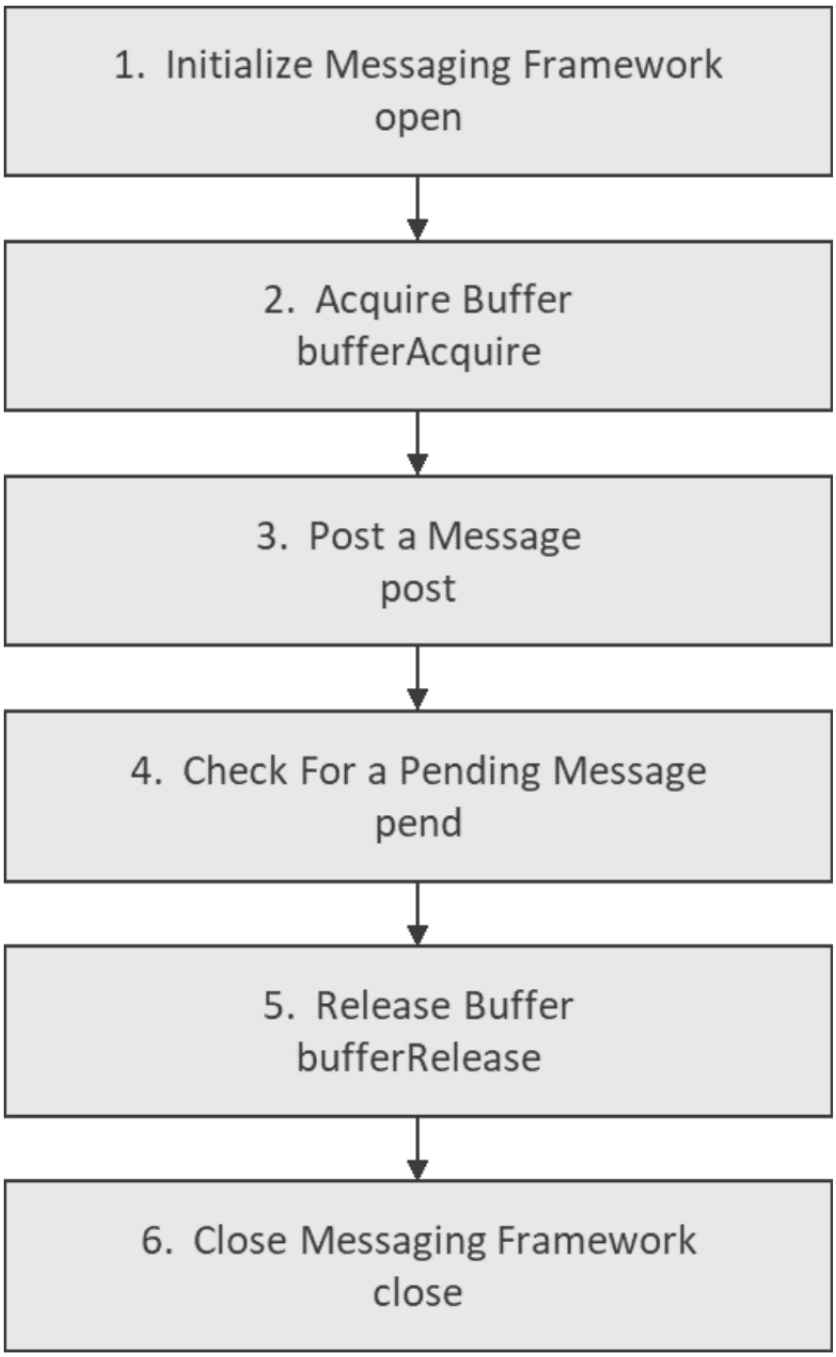


Figure 197: Flow Diagram of a Typical Messaging Framework Module Application

4.1.23 Power Profiles V2 Framework

4.1.23.1 Power Profiles V2 Framework Introduction

The Power Profiles V2 Framework provides a high-level API used to control the system clocks, the I/O ports, the operating modes (indirectly through the clock control) and the low power modes of the

MCU. The Power Profiles V2 Framework, when used with the LPM V2 Driver, CGC Driver and I/O Port Driver, gives the user advanced control over the power consumption of the MCU.

Power Profiles V2 Framework Module Features

- Uses Low Power Modes V2
- Sets CGC clock configuration and I/O Port pin configuration when entering and exiting the configured low power mode
- Supports both threaded and non-threaded operations

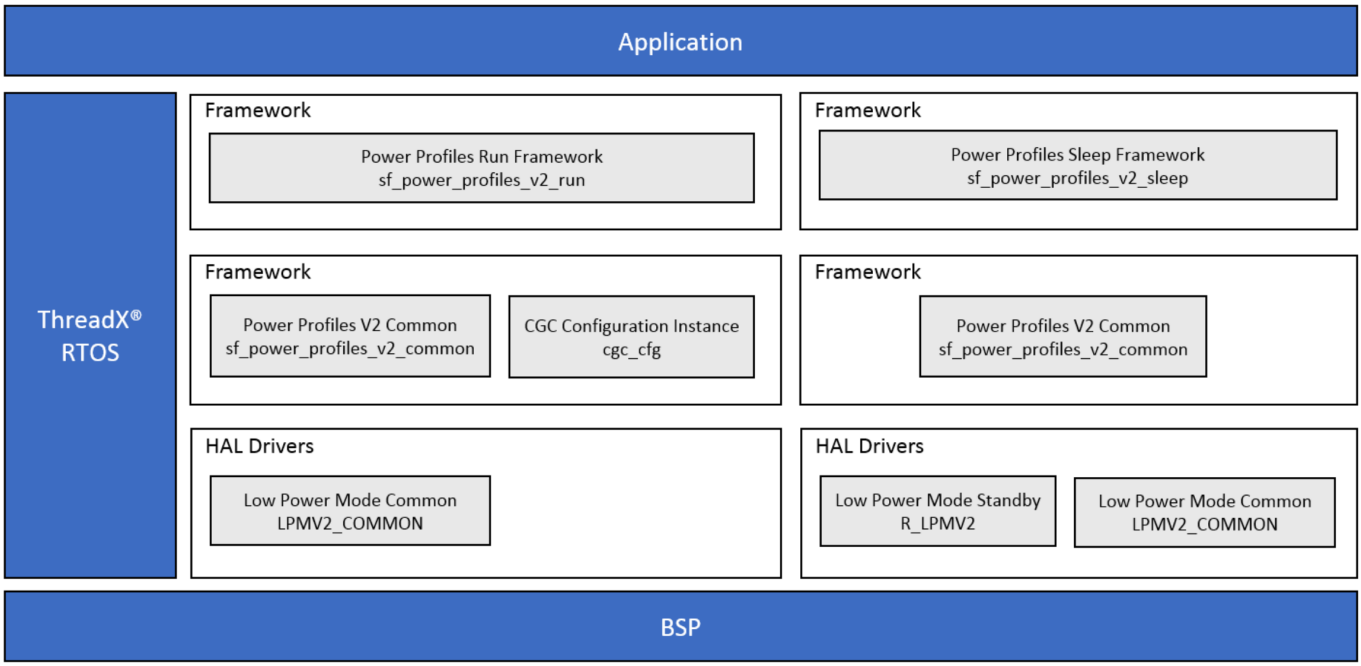


Figure 198: Power Profiles V2 Framework Module Block Diagram

4.1.23.2 Power Profiles V2 Framework Module APIs Overview

There are different low-level LPM V2 HAL modules used in the framework depending on the target MCU (as shown in the following table):

MCU	Driver
S124	S124 Low Power Mode Sleep on r_lpmv2
S124	S124 Low Power Mode Standby on r_lpmv2
S128	S128 Low Power Mode Sleep on r_lpmv2
S128	S128 Low Power Mode Standby on r_lpmv2
S3A3	S3A3 Low Power Mode Sleep on r_lpmv2
S3A3	S3A3 Low Power Mode Standby on r_lpmv2
S3A7	S3A7 Low Power Mode Sleep on r_lpmv2
S3A7	S3A7 Low Power Mode Standby on r_lpmv2
S5D9	S5D9 Low Power Mode Sleep on r_lpmv2

S5D9	S5D9 Low Power Mode Standby on r_lpmv2
S5D9	S5D9 Low Power Mode Deep Standby on r_lpmv2
S7G2	S7G2 Low Power Mode Sleep on r_lpmv2
S7G2	S7G2 Low Power Mode Standby on r_lpmv2
S7G2	S7G2 Low Power Mode Deep Standby on r_lpmv2

Because of the large number of lower level LPM V2 HAL modules available, it would be difficult to identify separate APIs and configuration settings. This module guide only goes into details for the S7G2 MCU. All the APIs and configuration settings are the same (except for those MCUs that do not support Deep Standby). It should be simple to extrapolate to any target MCU.

The Power Profiles defines APIs for common functions such as open, sleep and close. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Power Profiles V2 Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_power_profiles_v2_common.p_api->open(g_sf_power_profiles_v2_low_power_0.p_ctrl, g_sf_power_profiles0.p_cfg); Initialize the Power Profiles V2 Framework.
runApply	g_sf_power_profiles_v2_common.p_api->runApply(g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_run_0); Apply the selected run power profile.
lowPowerApply	g_sf_power_profiles_v2_common.p_api->lowPowerApply(g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_low_power_0); Apply the selected low power profile.
close	g_sf_power_profiles_v2_common.p_api->close(g_sf_power_profiles_v2_common.p_ctrl); Close the module.
versionGet	g_sf_power_profiles_v2_common.p_api->versionGet(&p_version); Get version and store it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description

SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	The framework has already been initialized.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.
SSP_ERR_NOT_OPEN	Device not open.
SSP_ERR_UNSUPPORTED	The function is not supported by the module.
SSP_ERR_INTERNAL	Internal error.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.23.3 Power Profiles V2 Framework Module Operational Overview

The Power Profiles V2 Framework provides a high-level API used to control the system clocks, the I/O ports, the operating modes (indirectly through the clock control), and the low power modes of the MCU. The Power Profiles V2 Framework, when used with the LPM V2 Driver, CGC Driver, and I/O Port Driver, gives the user advanced control over the power consumption of the MCU.

The Power Profiles V2 Framework provides 2 main functions to control the MCU power consumption, sf_power_profiles_v2_api_t::runApply and sf_power_profiles_v2_api_t::lowPowerApply. The runApply() function uses a CGC Clocks configuration and an I/O Port pin configuration to set the system clocks and I/O Port pins of the MCU. The lowPowerApply() function uses a LPM V2 configuration and two I/O Port configuration to set the low power mode and I/O Port pins before entering the configured low power mode and after waking from the low power mode. See LPM V2 module overview and the MCU hardware manual for details about the available low power modes.

The Power Profiles V2 Framework uses the LPM V2, IOPORT, and CGC Drivers of the Synergy Software Package and provides an easy-to-use software interface to control the power consumption of the MCU.

Operational Description

The Power Profiles V2 Framework configures the system in both a Run state and a Low Power state. The system clocks, I/O pins, and low power mode of the MCU can all be handled and controlled using the Power Profiles V2 Framework.

The Power Profiles V2 Framework API function sf_power_profiles_v2_api_t::open initializes the Power Profiles V2 Framework internal variables, instance variables, and the LPM V2 Driver. If the project uses ThreadX, the Framework will ensure safe use in a multi-threaded environment.

The runApply() and lowPowerApply() functions optionally use I/O Port pin configurations to provide control of the MCU I/O Ports. The lowPowerApply() function can use 2 pin configurations: one to set the pins to a state appropriate for the low power mode when the MCU will not be executing instructions, and a second for after waking from the low power mode, when instruction execution resumes.

The Power Profiles V2 Framework runApply() function applies the user-defined optional pin configuration, then applies the user-defined CGC clocks configuration. The user can switch clocks on and off, change clock dividers, and switch the system clocks using the CGC Clocks configuration structure.

The Power Profiles V2 Framework API function runApply() performs the following tasks in order:

1. If the project uses ThreadX, the function will get the ThreadX mutex prior to calling any lower level driver.
2. Apply the user-specified optional pin configuration.
3. Apply the user-specified CGC Clocks configuration.
4. If the project uses ThreadX, the function will return the ThreadX mutex.

The lowPowerApply() function uses an LPM V2 configuration to set the low power mode, the triggers for waking from the low power mode, the state of bus pins, and other settings that are MCU specific. The lowPowerApply() function can optionally use an application callback function. The prototype can be found in /src/synergy_gen/hal_data.c or /src/synergy_gen/<thread name>.c.

The Power Profiles V2 Framework lowPowerApply() function performs the following tasks in order:

1. If the project uses ThreadX, the function will get the ThreadX mutex prior to calling any lower level driver.

2. Apply the optional user-specified low power entry pin configuration.

3. Call the user specified callback function with the enumeration SF_POWER_PROFILES_V2_EVENT_PRE_LOW_POWER.

4. Apply the user-specified low power mode configuration. Any valid LPM V2 configuration can be used.

5. Enter the low power mode.

6. If the low power mode chosen was other than Deep Standby the MCU will resume execution of code from the same point once the wakeup trigger is detected. (If the LPM V2 low power mode configuration was Deep Standby, the MCU will not resume code execution but will instead go through a soft reset once the wakeup trigger is detected.)

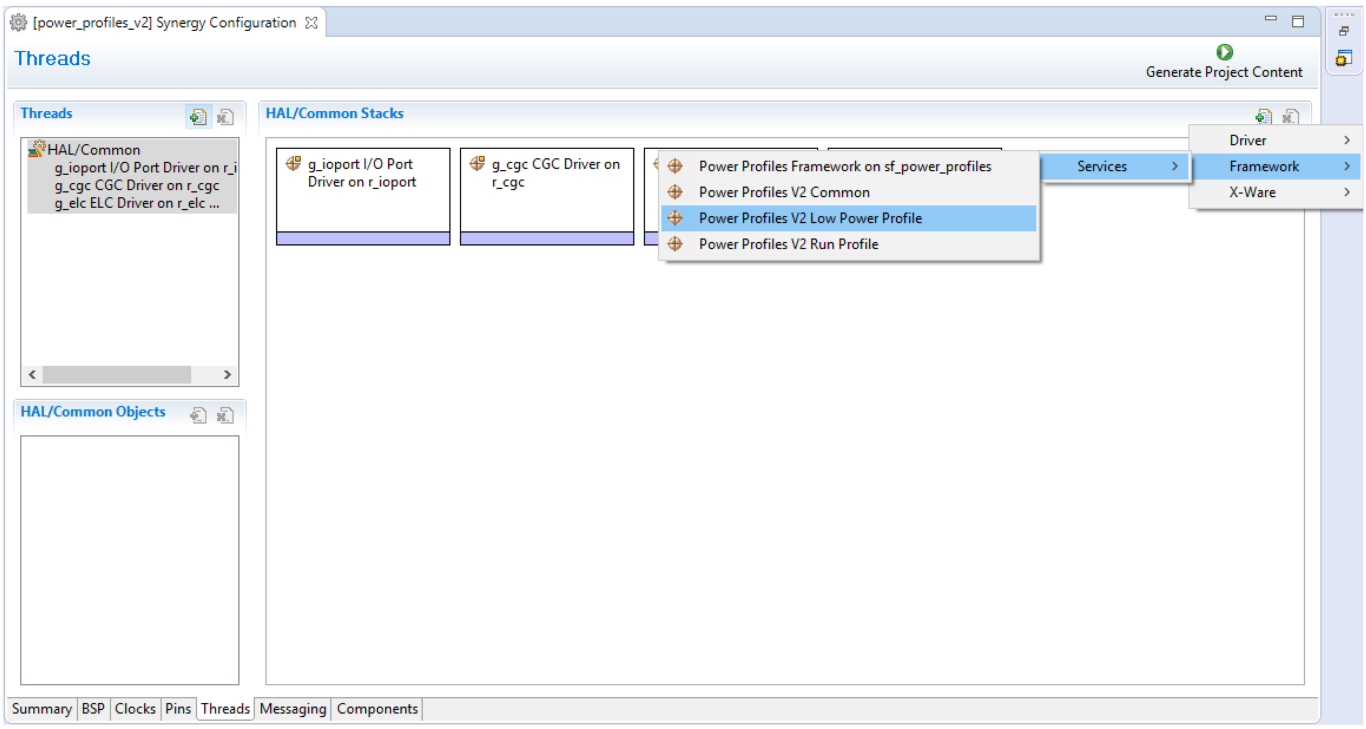
7. Apply the optional user-specified low power exit pin configuration.

8. Call the user specified callback function with the enumeration SF_POWER_PROFILES_V2_EVENT_POST_LOW_POWER.

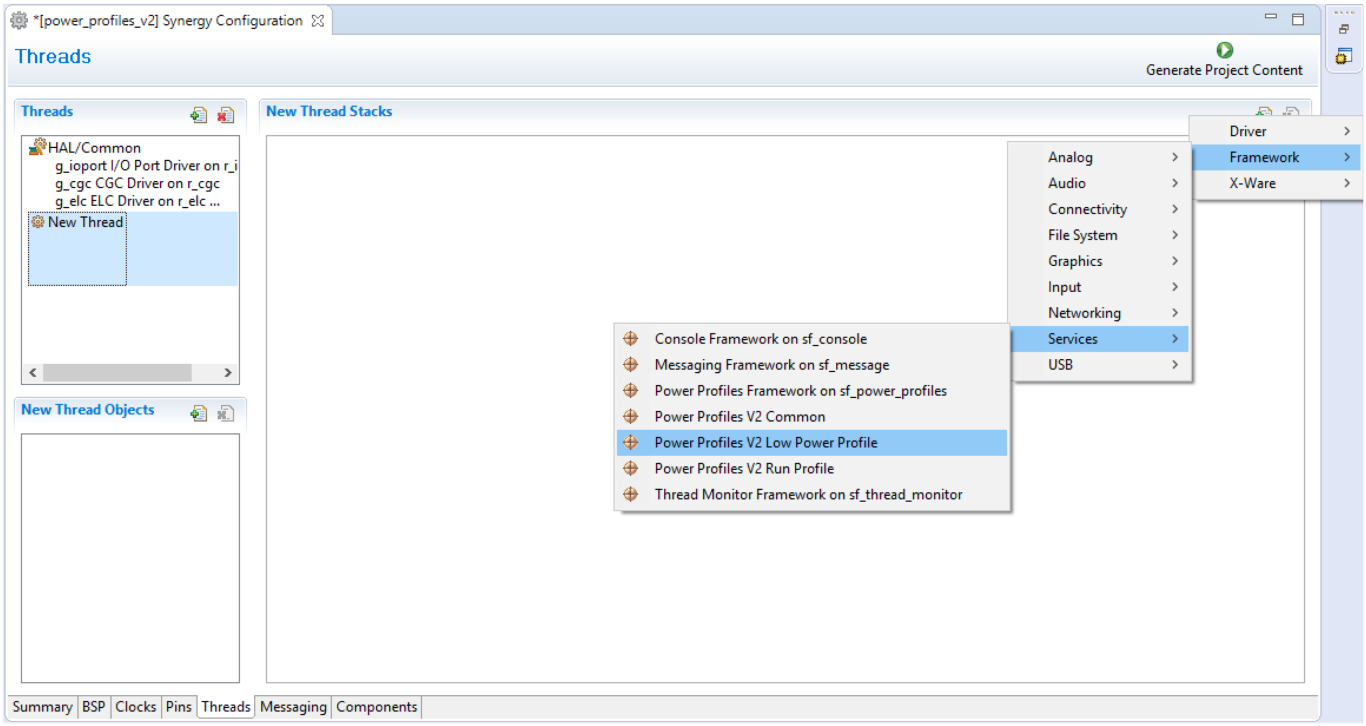
9. If the project uses ThreadX, the function will return the ThreadX mutex.

The Power Profiles V2 Run or Low Power profiles should be added to the project by the user. The Power Profiles V2 common module will be added automatically.

Outside of a thread:

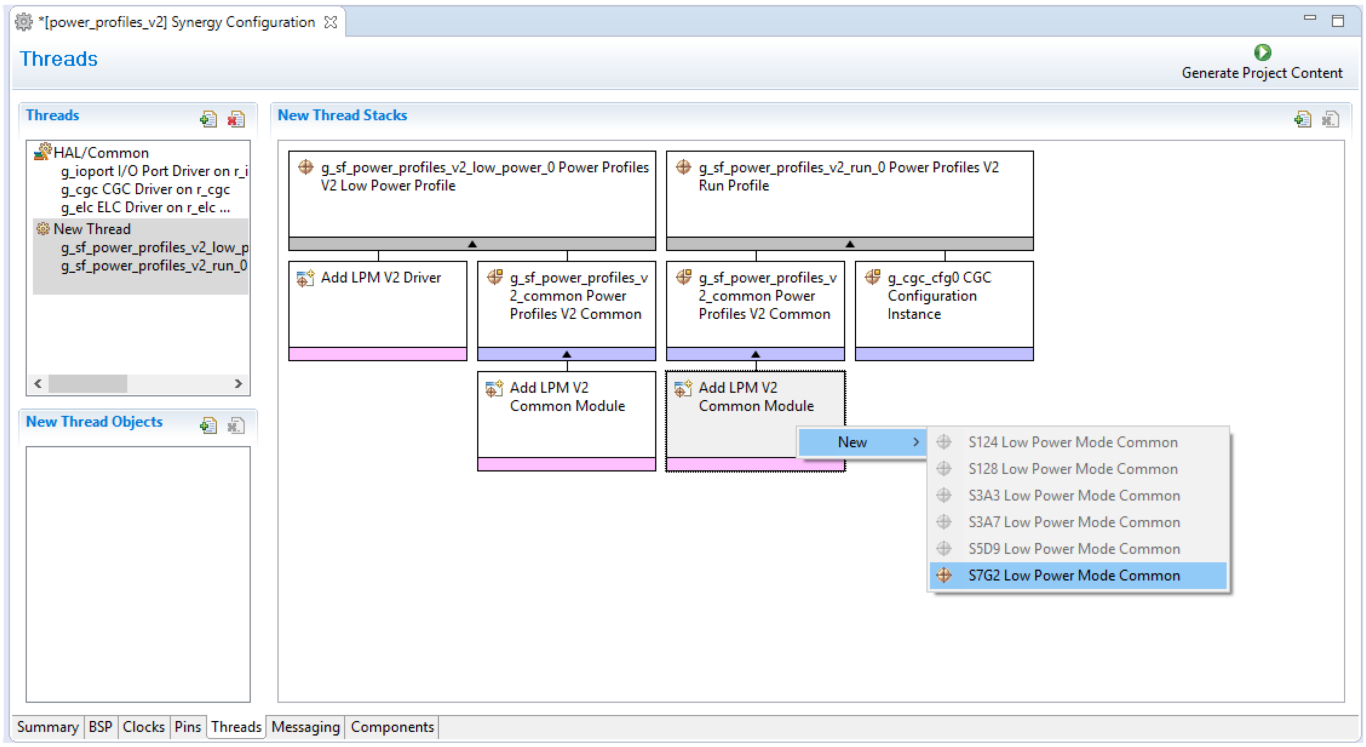


Within a thread:

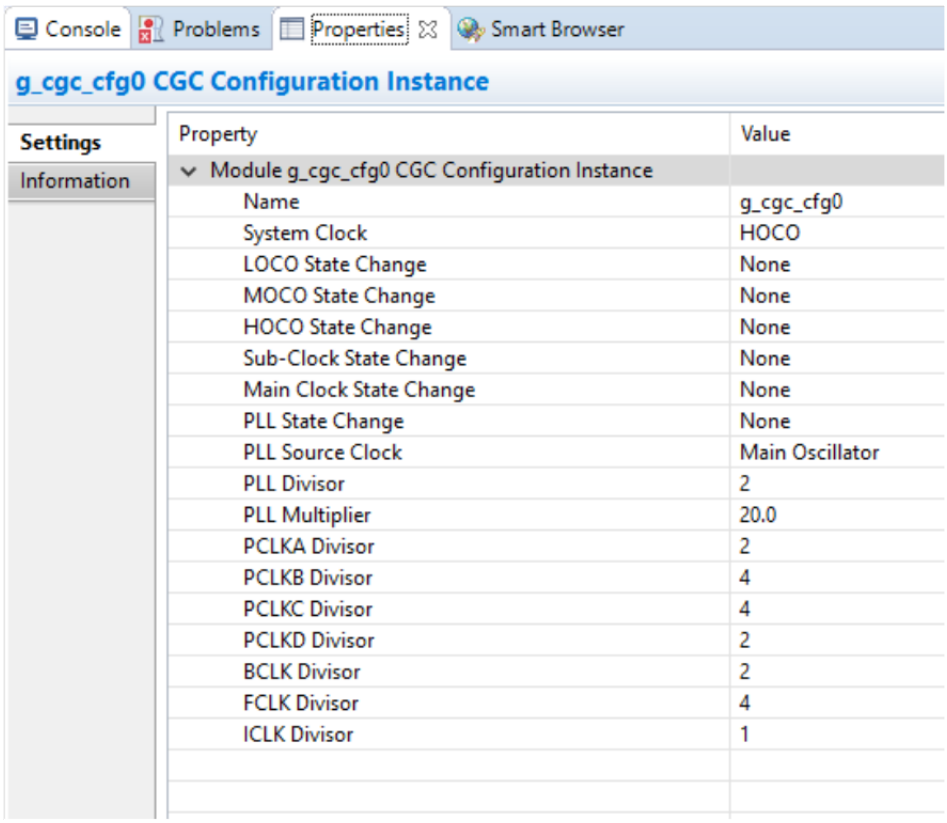


After adding a Power Profiles V2 Run or Low Power Profile, you will need to add an LPM V2 Common

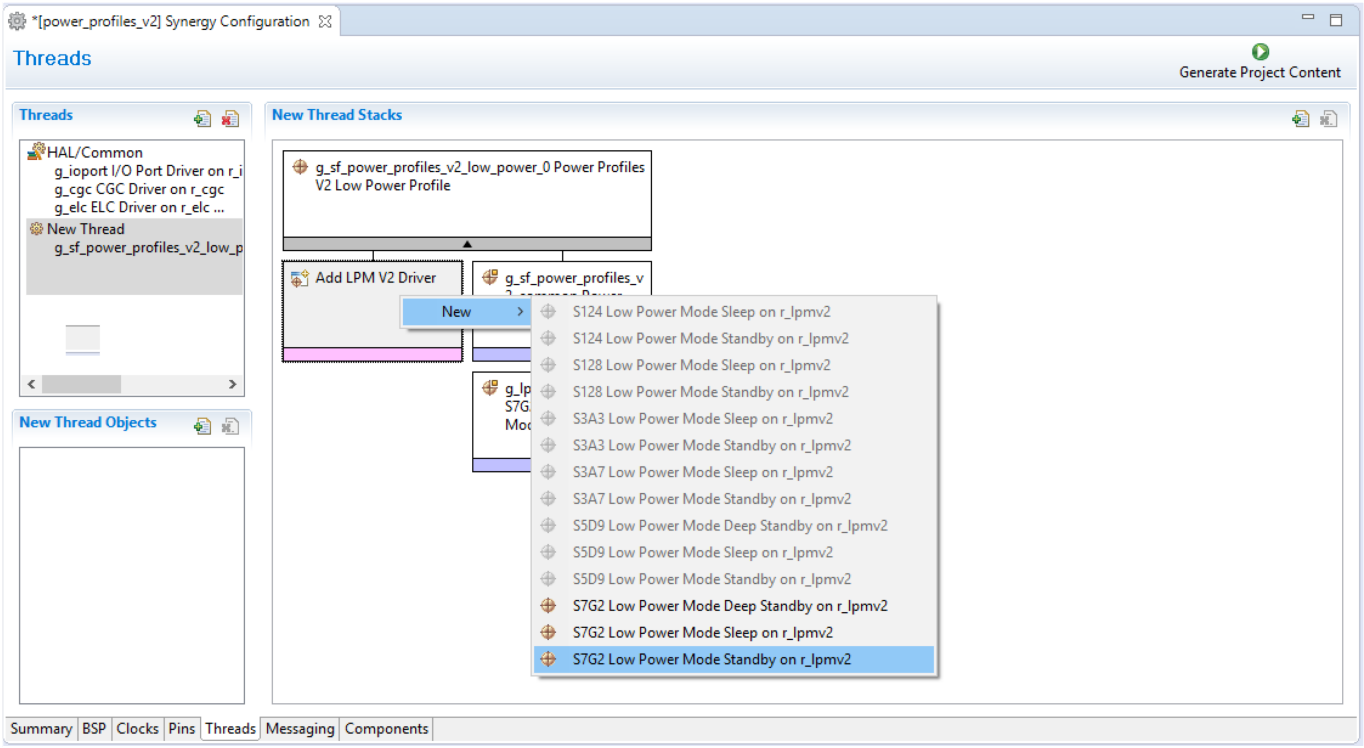
module. If an LPM V2 Common instance already exists in the project, it will be used. A Power Profiles V2 Run Profile does not directly use the LPM V2, but it is still a dependency for a successful build.



A Power Profiles V2 Run Profile depends upon a CGC Clocks Configuration. The configuration options of the CGC Clocks Configuration instance are provided below. If a CGC Clocks Configuration instance is not already present in the project, it will be automatically added. If one is present, it could be used instead. Controlling system clocks is a critical part of controlling power consumption of an MCU. See the CGC usage notes for more information.



The Power Profiles V2 Low Power Profile uses an LPM V2 standby instance, but an LPM V2 deep standby instance or LPM V2 sleep instance could be used instead depending on which MCU is currently being used.



The LPM V2 instance's properties can be configured by selecting the instance and reviewing the Properties pane.

Configuring the Pins for the Power Profiles V2 Framework

Optional: Create additional I/O Port pin configurations using the **Pins** tab as follows:

1. In the top-level directory of your project, find the file with file extension *.pincfg.

2. Make a copy of this file and rename it, keeping the new file in the same top-level directory of the project.

3. The new file is now available as an option on the Pins tab of the Synergy Configuration. Look for the file name in the drop down list on the Pins tab below "Select pin configuration".

4. Check the Generate data checkbox and type in a pin configuration name. The checkbox and text entry can be found to the right of the pin configuration drop down.

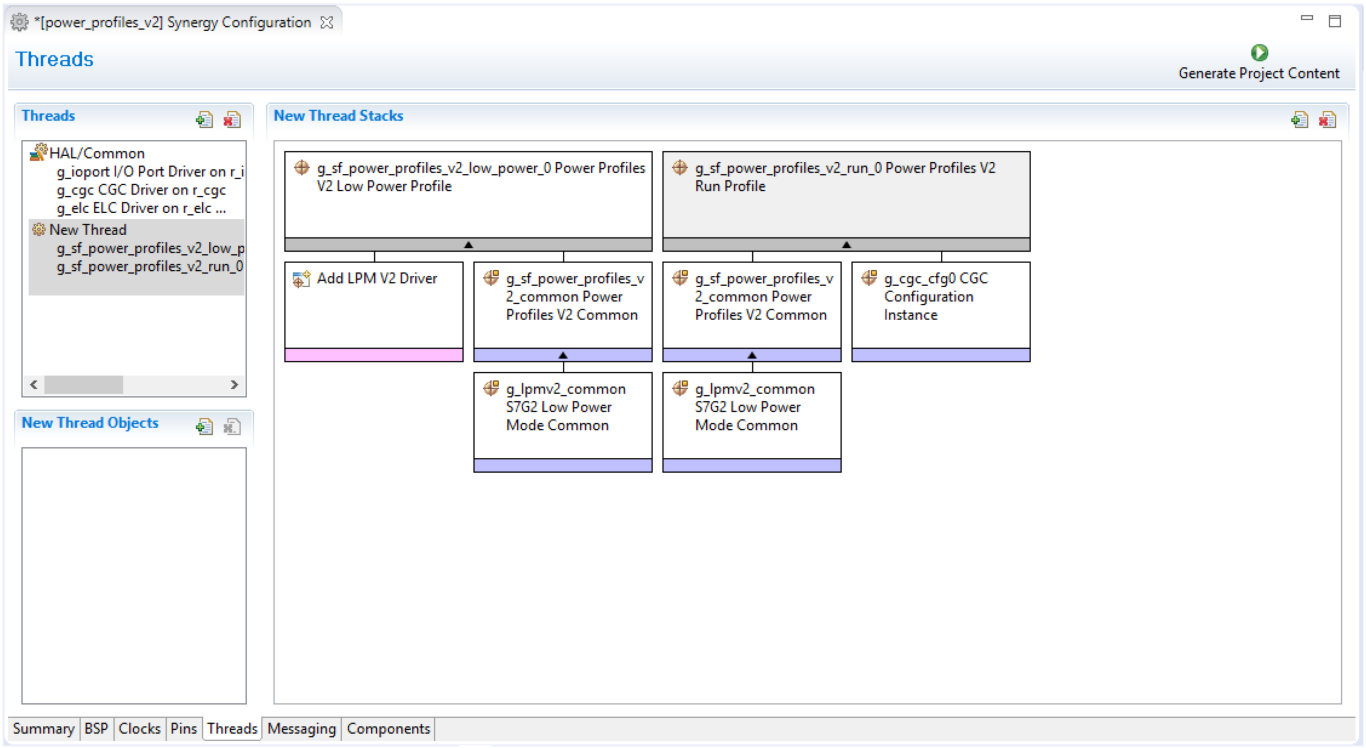
5. Configure the pins as desired for either a Run or Low Power Profile.

6. Save the project configuration and Generate Project Content.

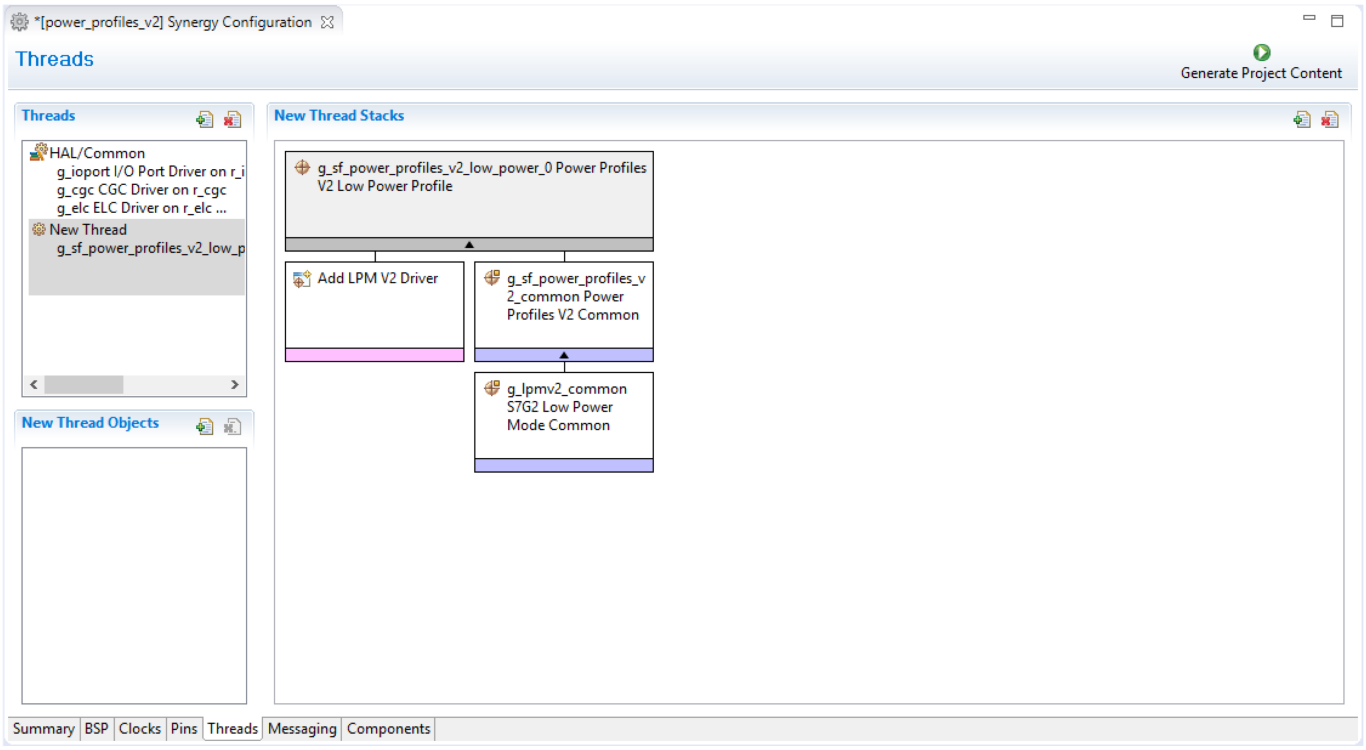
7. To view the pin configuration that was generated, look in the file {project_directory}/src/synergy_gen/pin_data.c for the ioport_cfg_t structure of the same name as entered in the text box.

8. Add the name of the ioport_cfg_t structure to one of the pin configuration table entries for a Run or Low Power Profile.

For a Power Profiles V2 Run Profile:



For a Power Profiles V2 Low Power Profile:



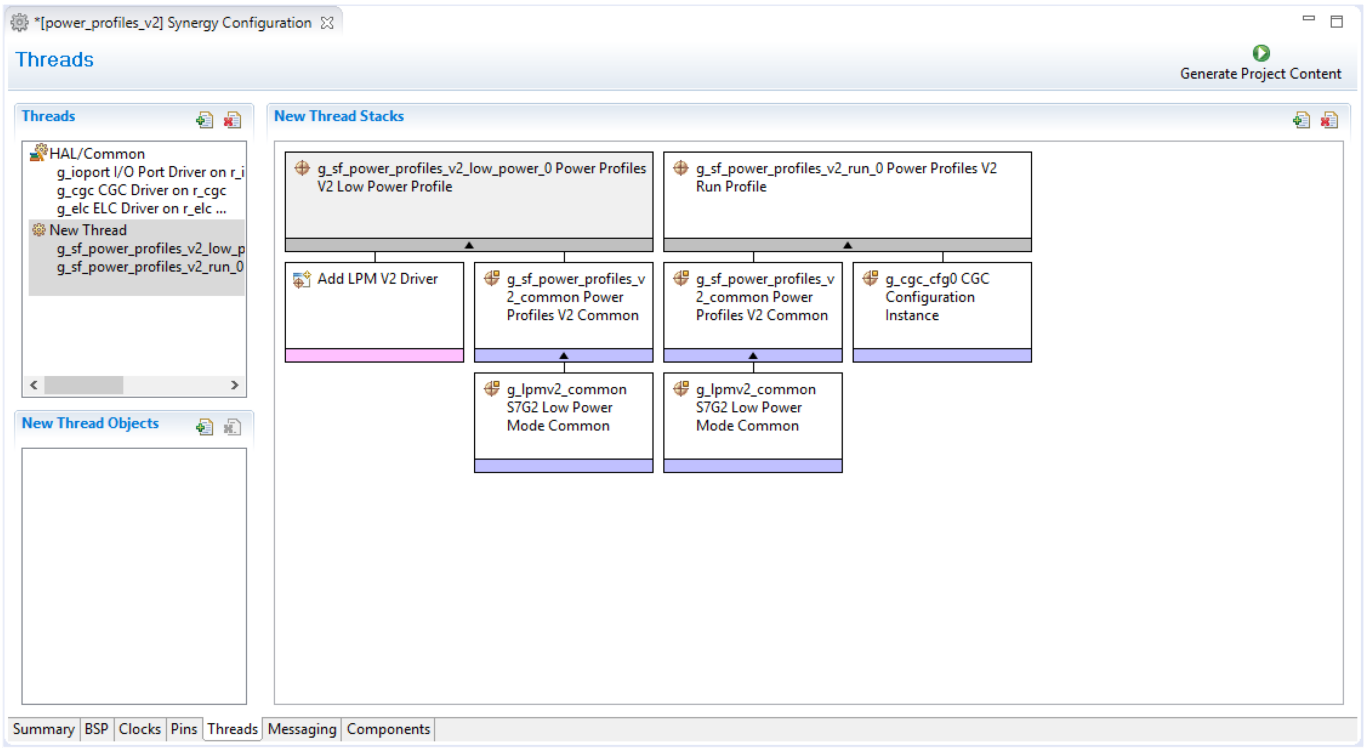
Configuring the Interrupts for the Power Profiles V2 Framework

The Power Profiles V2 Framework does not use any interrupts directly, although any interrupt is capable of waking the MCU while in Sleep mode. This is handled through the LPM V2 driver configuration.

Configuring the Power Profiles V2 Callbacks

Power Profiles V2 Low Power Profiles can notify the application before entering the low power mode and after waking from low power mode. The prototype can be found in /src/hal_data.c or /src/synergy_gen/<thread name>.c.

Fill in the callback used by the Power Profiles V2 Low Power Profile:



Configuring the Low Power Module Parameters

See the LPM V2 usage notes for an in depth description of how to use LPM V2.

Power Profiles V2 Framework Module Important Operational Notes and Limitations

Power Profiles V2 Framework Module Operational Notes

- An LPM V2 Driver instance is required to create Power Profiles V2 applications. The CGC driver is included in a Synergy project by default. To use the Power Profiles V2 runApply() function, another instance of the CGC Driver is required in order to create a CGC Clocks configuration. Since the CGC Driver is included in all Synergy projects by default, this does not add to the code size of the project.
- The I/O Port pin configurations can be created without adding an additional instance of the I/O Port driver.

- When used with ThreadX, this framework uses ThreadX intrinsic objects like mutexes. Operation with ThreadX is optional.
- Power Profiles V1 and Power Profiles V2 cannot be used in the same project. For all new projects, it is recommended that applications use Power Profiles V2.

Power Profiles V2 Framework Module Limitations

The Power Profiles V2 Framework does not handle starting or stopping MCU peripherals.

The Power Profiles V2 Framework open function will not be called automatically prior to main if the project does not use ThreadX. The initialization must be done explicitly by calling g_common_init() or by explicitly calling the sf_power_profiles_v2_api_t::open API. This is not a Power Profiles V2 limitation but a result of any Framework module that supports being used without an RTOS.

```
#include "hal_data.h"
void hal_entry(void)
{
    g_common_init();
    g_sf_power_profiles_v2_common.p_api->runApply(
        g_sf_power_profiles_v2_common.p_ctrl,
        &g_sf_power_profiles_v2_run_0);
    g_sf_power_profiles_v2_common.p_api->lowPowerApply(
        g_sf_power_profiles_v2_common.p_ctrl,
        &g_sf_power_profiles_v2_low_power_0);
    g_sf_power_profiles_v2_common.p_api->close(g_sf_power_profiles_v2_common.p_ctrl);
}
```

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.23.4 Including the Power Profiles V2 Framework Module in an Application

This section describes how to include the Power Profiles V2 Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Power Profiles V2 Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Power Profiles V2 Framework module is sf_power_profiles_v2_0. This name can be changed in the associated Properties window.)

Power Profiles V2 Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_power_profiles_low_power_0 Power Profiles V2 Low Power Profile	Threads	**New Stack> Framework> Services> Power Profiles V2 Low Power Profile**
g_sf_power_profiles_run_0 Power Profiles V2 Run Profile	Threads	**New Stack> Framework> Services> Power Profiles V2 Run Profile**

When the Power Profiles V2 Framework module on sf_power_profiles_v2 is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

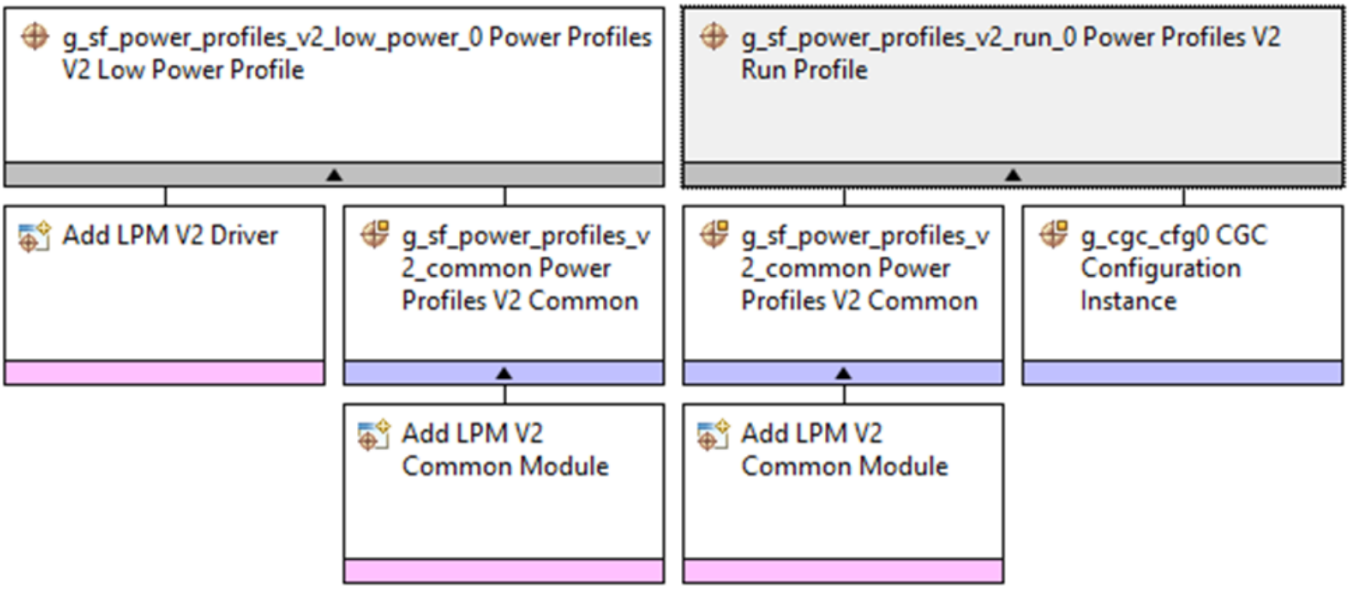


Figure 199: Power Profiles V2 Framework Module Stack

4.1.23.5 Configuring the Power Profiles V2 Framework Module

The Power Profiles V2 Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

There are two different stack selections for the Power Profiles V2 Framework, the Run Profile and the Low Power Profile. Their respective configuration settings will be covered separately in the following sections.

Configuring the Power Profiles V2 Run Profile

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the Power Profiles V2 Run Profile

ISDE Property	Value	Description
Name	g_sf_power_profiles_v2_run_0	Module name.
Pin configuration table	NULL	Pin configuration table selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Power Profiles V2 Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_power_profiles_v2_common	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the CGC Configuration Instance

ISDE Property	Value	Description
Name	g_cgc_cfg0	Module name.
System Clock	HOCO, MOCO, LOCO, Main Oscillator, Sub Clock, PLL Default: HOCO	System clock selection.

LOCO State Change	None, Start, Stop Default: None	LOCO state change selection.
MOCO State Change	None, Start, Stop Default: None	MOCO state change selection.
HOCO State Change	None, Start, Stop Default: None	HOCO state change selection.
Sub-Clock State Change	None, Start, Stop Default: None	Sub-clock state change selection.
Main Clock State Change	None, Start, Stop Default: None	Main clock state change selection.
PLL State Change	None, Start, Stop Default: None	PLL state change selection.
PLL Source Clock	HOCO, MOCO, LOCO, Main Oscillator, Sub Clock, PLL Default: HOCO	PLL source clock selection.
PLL Divisor	1, 2, 3, 4 Default: 1	PLL divisor seiection.
PLL Multiplier	10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5, 15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5, 20.0, 20.5, 21.0, 21.5, 22.0, 22.5, 23.0, 23.5, 24.0, 24.5, 25.0, 25.5, 26.0, 26.5, 27.0, 27.5, 28.0, 28.5, 29.0, 29.5, 30.0, 31.0 Default: 10.0	PLL multiplier selection.
PCLKA Divisor	1, 2, 4, 8, 16, 64 Default: 1	PCLKA divisor selection.
PCKLB Divisor	1, 2, 4, 8, 16, 64 Default: 1	PCKLB divisor selection.
PCLKC Divisor	1, 2, 4, 8, 16, 64 Default: 1	PCLKC divisor selection.

PCLKD Divisor	1, 2, 4, 8, 16, 64 Default: 1	PCLKD divisor selection.
BCLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	BCLK divisor selection.
FCLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	FCLK divisor selection.
ICLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	ICLK divisor selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Lower Power Mode Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_common	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Power Profiles V2 Low Power Profile

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Power Profiles V2 Low Power Profile

ISDE Property	Value	Description
Name	g_sf_power_profiles_v2_low_power_0	Module name.
Callback (Low Power Exit Event N/A when using Deep Software Standby)	NULL	Callback selection.
Low power entry pin configuration table	NULL	Low power entry pin configuration table selection.

Low power exit pin configuration table	NULL	Low power exit pin configuration table selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Low Power Mode Deep Standby

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_deep_standby	Module name.
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection.
Maintain or reset the IO port states on exit from deep standby mode	Maintain the IO port states, Reset the IO port states Default: Maintain the IO port states	Maintain or reset the IO port states selection.
Internal power supply control in deep standby mode	Maintain the internal power supply, Cut the power supply to standby RAM, low-speed on-chip oscillator, AGTn, and USPFS/HS resume detecting unit, Cut the power supply to LVDn, standby RAM, low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit Default: Maintain the internal power supply	Internal power supply control selection.
IRQ0-15	Enabled, Disabled Default: Disabled	IRQ0-15 selection.
IRQ0-15 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	IRQ0-15 Edge selection.
LVD1	Enabled, Disabled Default: Disabled	LVD1 selection.

LVD1 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD1 Edge selection.
LVD2	Enabled, Disabled Default: Disabled	LVD2 selection.
LVD2 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD2 Edge selection.
RTC Interval	Enabled, Disabled Default: Disabled	RTC Interval selection.
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection.
NMI	Enabled, Disabled Default: Disabled	NMI selection.
NMI Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	NMI Edge selection.
USBFS	Enabled, Disabled Default: Disabled	USBFS selection.
UBSHS	Enabled, Disabled Default: Disabled	UBSHS selection.
AGT1	Enabled, Disabled Default: Disabled	AGT1 selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Low Power Mode Sleep

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_sleep0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Low Power Mode Standby

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_standby0	Module name.
Choose the low power mode	Standby, Standby with snooze Enabled Default: Standby	Low power mode selection.
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection.
IRQ1-15	Enabled, Disabled Default: Disabled	IRQ1-15 selection.
IWDT	Enabled, Disabled Default: Disabled	IWDT selection.
Key Interrupt	Enabled, Disabled Default: Disabled	Key Interrupt selection.
LVD1 Interrupt	Enabled, Disabled Default: Disabled	LVD1 Interrupt selection.
LVD2 Interrupt	Enabled, Disabled Default: Disabled	LVD2 Interrupt selection.
Analog Comparator High-speed 0 Interrupt	Enabled, Disabled Default: Disabled	Analog Comparator High-speed 0 Interrupt selection.
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection.
RTC Period	Enabled, Disabled Default: Disabled	RTC Period selection.
USB High-speed	Enabled, Disabled Default: Disabled	USB High-speed selection.

USB Full-speed	Enabled, Disabled Default: Disabled	USB Full-speed selection.
AGT1 underflow	Enabled, Disabled Default: Disabled	AGT1 underflow selection.
AGT1 Compare Match A	Enabled, Disabled Default: Disabled	AGT1 Compare Match A selection.
AGT1 Compare Match B	Enabled, Disabled Default: Disabled	AGT1 Compare Match B selection.
12C 0	Enabled, Disabled Default: Disabled	12C 0 selection.
Snooze Entry Source	RXD0 falling edge, IRQ0-IRQ15, KINT, ACMPHS0, RTC Alarm, RTC Period, AGT1 Underflow, AGT1 Compare Match A, AGT1 Compare Match B Default: RXD0 falling edge	Snooze Entry Source selection.
AGT1 Underflow	Enabled, Disabled Default: Disabled	AGT1 Underflow selection.
DTC Transfer Completion	Enabled, Disabled Default: Disabled	DTC Transfer Completion selection.
DTC Transfer Completion Negated Signal	Enabled, Disabled Default: Disabled	DTC Transfer Completion Negated Signal selection.
ADC0 Compare Match	Enabled, Disabled Default: Disabled	ADC0 Compare Match selection.
ADC0 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC0 Compare Mismatch selection.
ADC1 Compare Match	Enabled, Disabled Default: Disabled	ADC1 Compare Match selection.
ADC1 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC1 Compare Mismatch selection.
SCI0 Address Match	Enabled, Disabled Default: Disabled	SCI0 Address Match selection.

DTC state in Snoooze Mode	Enabled, Disabled Default: Disabled	DTC state in Snoooze Mode selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Power Profiles V2 Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_power_profiles_v2_common	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Low Power Mode Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_common	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Power Profiles V2 Framework Module Clock Configuration

The Power Profiles V2 framework does not require any specific clock settings

Power Profiles V2 Framework Module Pin Configuration

The application may optionally maintain the I/OPort state during a low power mode.

4.1.23.6 Using the Power Profiles V2 Framework Module in an Application

The typical steps in using the Power Profiles V2 Framework module in an application are:

1. Define the body of the callback function configured in the Low Power profile. The callback function notifies the application when the MCU is about to enter a low power mode and when the MCU just woke up from a low power mode. Using a callback is optional, but if you define a callback in the Power Profiles V2 properties, then there must be a definition for it.
2. If ThreadX is used, the Power Profiles V2 Framework sf_power_profiles_v2_api_t::open function will be called by the Synergy generated code before the user application code is

reached. If ThreadX is not used, the application must call the open function.

3. Apply a Run Profile at any time using the runApply() function. The runApply() function accepts a Run Profile as its second parameter. The parameter can be any valid Run Profile, allowing the application to easily switch between Run Profiles. g_sf_power_profiles_v2_common.p_api->runApply(g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_run_0);
4. Apply a Low Power Profile using the lowPowerApply() function. The lowPowerApply() function accepts a Low Power Profile as its second parameter. The parameter can be any valid Low Power Profile, allowing the application to easily switch between Low Power Profiles g_sf_power_profiles_v2_common.p_api->lowPowerApply(g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_low_power_0);
5. Close the framework by calling the sf_power_profiles_v2_api_t::close function. [Optional] g_sf_power_profiles_v2_common.p_api->close(g_sf_power_profiles_v2_common.p_ctrl);

These common steps are illustrated in a typical operational flow in the following figure:

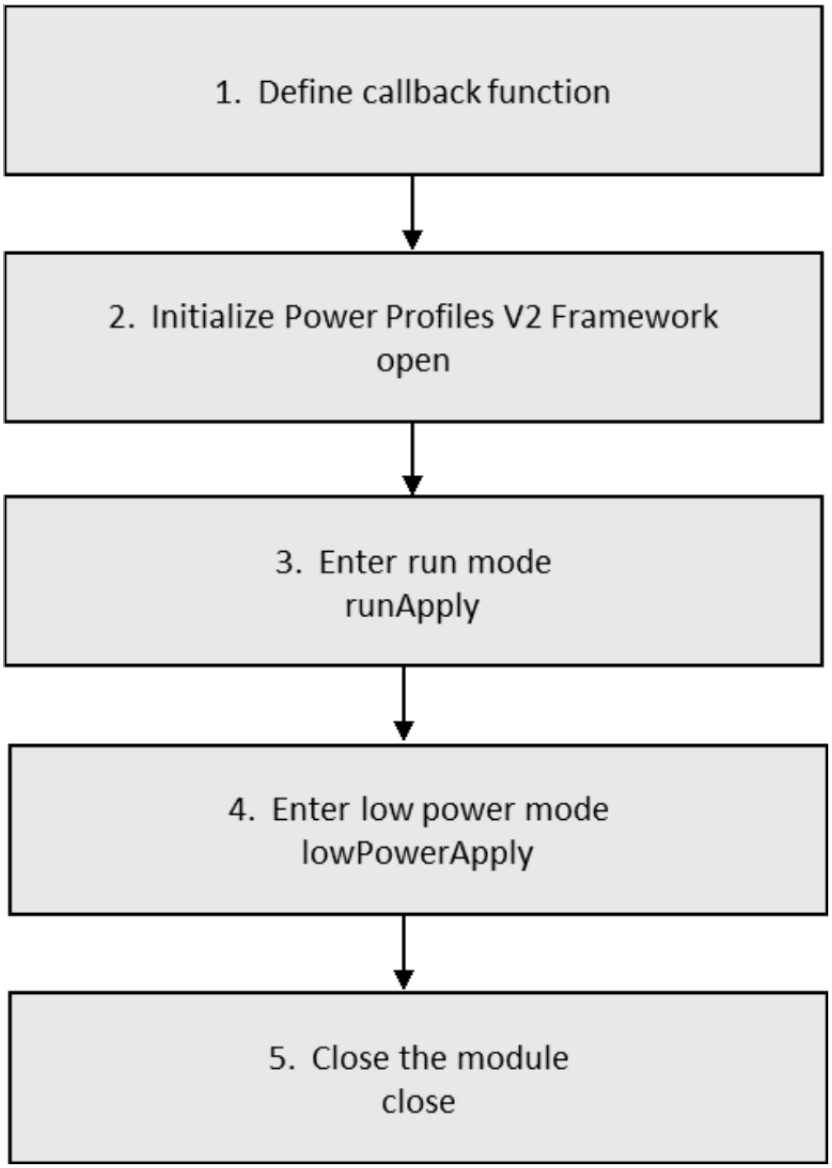


Figure 200: Flow Diagram of a Typical Power Profiles V2 Framework Module Application

4.1.24 SPI Framework

4.1.24.1 SPI Framework Introduction

The SPI Framework module provides a ThreadX-aware framework API and handles the integration and synchronization of multiple SPI peripherals on an SPI bus (including chip-select handling and its level activation). With the SPI Framework, one or more SPI buses can be created and multiple SPI peripherals can be connected to the SPI bus. The SPI Framework module uses a single interface to access both SCI SPI and RSPI drivers. The SPI Framework module uses the SCI and RSPI peripherals on the Synergy MCU.

SPI Framework Module Features

The SPI Framework module uses either the SCI in SPI mode (together with the SCI common lower-level modules) or the RSPI lower-level driver module to communicate with the SPI peripherals on the Synergy microcontroller.

- Supports multiple devices on a bus
- Provides high-level APIs for initialization, transfers and closing the module
- Supports synchronized transfers
- Supports chip-select operations
- Supports bus-locking

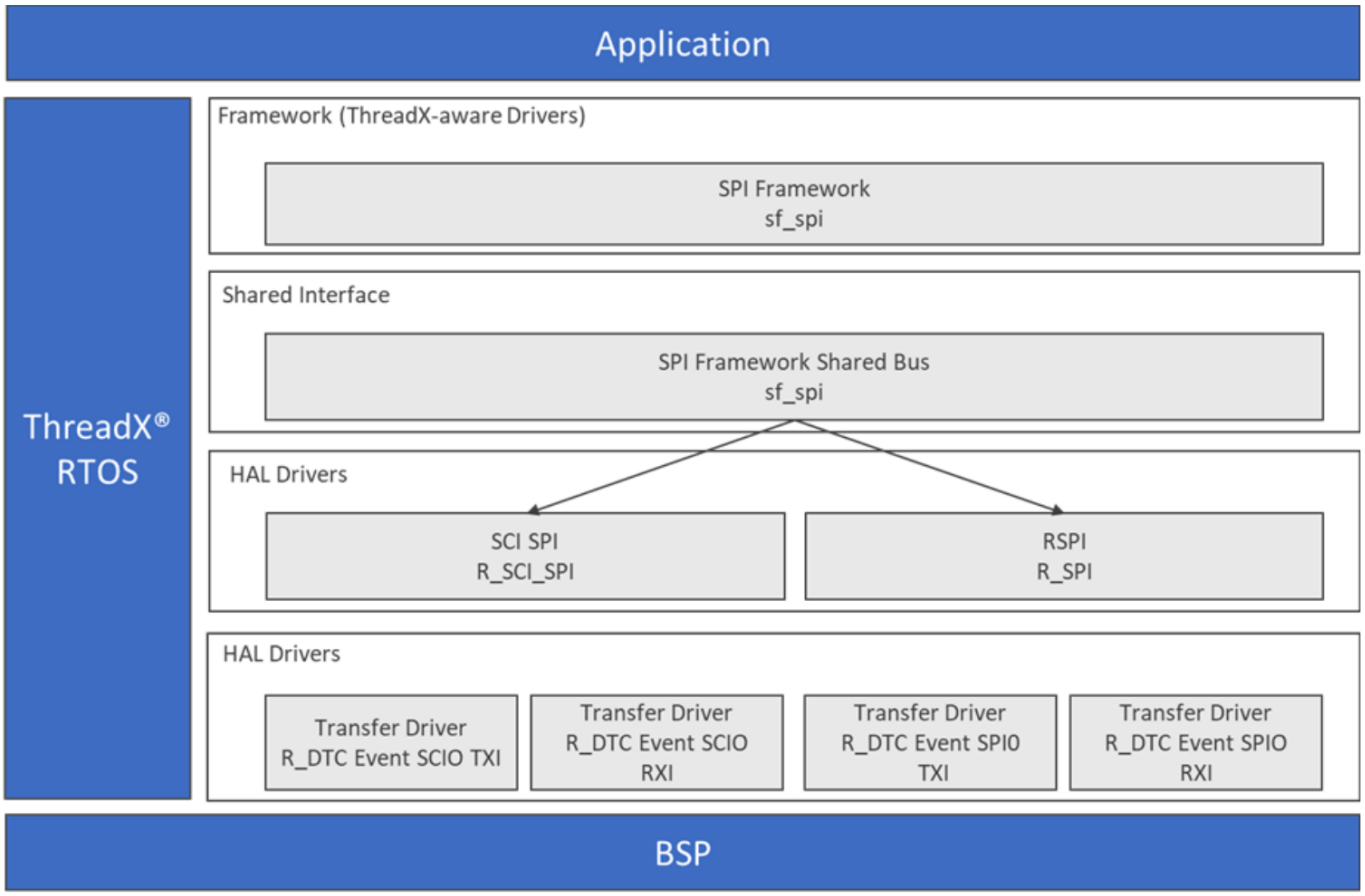


Figure 201: SPI Framework Module Block Diagram

4.1.24.2 SPI Framework Module APIs Overview

The SPI Framework module defines APIs for opening, closing, reading, writing and other useful functions. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SPI Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_spi_device.p_api->open(g_sf_spi_device.p_ctrl, g_sf_spi_device.p_cfg); Open a designated SPI device on a bus.
read	g_sf_spi_device.p_api->read(g_sf_spi_device.p_ctrl, &destination, length, SPI_BIT_WIDTH_8_BITS, timeout); Receive data from SPI device.
write	g_sf_spi_device.p_api->write(g_sf_spi_device.p_ctrl, &source, length, SPI_BIT_WIDTH_8_BITS, timeout); Transmit data to SPI device.
writeRead	g_sf_spi_device.p_api->writeRead (g_sf_spi_device.p_ctrl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, timeout); Simultaneously transmits data to an SPI device while receiving data from an SPI device (full duplex). The writeread API gets a mutex object, handles the SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer.
close	g_sf_spi_device.p_api->close(g_sf_spi_device.p_ctrl); Disable the SPI device designated by the control handle and close the RTOS services used by the bus, if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
lock	g_sf_spi_device.p_api->lock(g_sf_spi_device.p_ctrl); Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (such as between lock and unlock). This allows devices to complete several reads and writes on the bus without an interrupt.
unlock	g_sf_spi_device.p_api->unlock(g_sf_spi_device.p_ctrl); Unlock the bus for a particular device and make the bus usable for other devices.

version	g_sf_spi_device.p_api->version (&version); Retrieve the API version with the version pointer.
lockWait	g_sf_spi_device.p_api->lockWait(g_sf_spi_device.p_ctrl, timeout); Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt. The wait option allows thread to wait for the specified timeout when acquiring the bus mutex.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully.
SSP_ERR_INVALID_MODE	Invalid mode.
SSP_ERR_INVALID_CHANNEL	Invalid channel.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_INVALID_ARGUMENT	Invalid argument.
SSP_ERR_QUEUE_UNAVAILABLE	Queue unavailable.
SSP_ERR_INVALID_POINTER	Invalid pointer.
SSP_ERR_INTERNAL	Internal error.
SSP_ERR_TRANSFER_ABORTED	Transfer aborted.
SSP_ERR_MODE_FAULT	Mode fault.
SSP_ERR_READ_OVF	Read overflow.
SSP_ERR_PARITY	Parity error.
SSP_ERR_OVERRUN	Overrun error.
SSP_ERR_UNDEF	Unknown error.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.24.3 SPI Framework Module Operational Overview

The SPI Framework module complies with the layered-driver architecture of the SSP. It uses either the SCI on SPI module or the RSPI module to communicate with the SPI peripherals on the Synergy microcontroller.

Multiple Slave Devices on the Same Bus

The SPI framework module uses a bus and device on bus architecture. Only one device is configured to the lower level driver at a time, and the other devices are reconfigured upon a read or write operation as required. The lower level driver can only be reconfigured when the bus is not locked. Every slave device is linked to the bus to which it will be connected and shares the bus with all other slave devices.

The user must configure the SPI framework shared-bus and the lower-level SPI HAL layer for each SPI framework module connecting to the bus. The user can add the existing framework shared-bus module when configuring multiple devices on the same bus. Each SPI framework module must be configured with a unique name in the ISDE configurator.

A common start and stop procedure is used for all SPI data-transfer operations (spi_api_t::read, spi_api_t::write and spi_api_t::writeRead). During the start process, the SPI framework module checks whether reconfiguration is required. Chip select is asserted during the transfer-start process and de-asserted during the transfer-end process if the bus is not locked. The user must configure the chip-select IO pin and the chip-select active level.

Bus Locking

The SPI Framework module supports bus-locking functionality, meaning that the bus can be locked for a given slave peripheral. The locking allows slave devices to reserve a bus to themselves for the period between the lock and unlock commands. This allows devices to complete several reads and writes on the bus without interruption (which can be required in some situations). The chip select becomes active during lock and becomes inactive when unlocked. Writes and reads in between the lock and unlock do not alter the chip-select line.

SPI Framework Module Important Operational Notes and Limitations

SPI Framework Module Operational Notes

- Multiple SPI devices can be configured to share a common bus. Once the SPI Framework bus module is configured, different SPI peripherals (devices) can be connected to that bus.
- For each SPI device connected to the bus, one SPI HAL module (new or shared) and one SPI Framework device module must be added.
- User defined Callback is not required as it has been internally taken care by framework.
- Setting the interrupts to different priority levels could result in improper operation.
- In the SPI Framework configuration, the channel number given to this bus overrides the channel number given in the HAL module.
- Shared bus can be used by multiple slave devices with the respective configuration. The framework also handles mutual exclusion in lock and unlock APIs when multiple devices are using the same SPI channel.
- Lock functionality will be effective for devices from different threads. If multiple devices connected to the bus are from the same thread, the SPI bus will be locked for all devices from that thread. In such cases, even if the bus is locked, all devices from the same thread can access the bus.
- In case a device used from multiple threads, and the device locks the SPI bus from one thread, the same device cannot access the SPI bus from other threads.

- The behavior of chip-select pin depends on the slave device. Chip-select pin can be utilized through framework or user can handle Chip-select pin in their application code based on the hardware specification of particular slave device.

SPI Framework Module Limitations

- Refer to the MCU specification manual for identifying SPI bus compatibility. Device compatibility with the SPI bus is not checked in the framework hence incompatible SPI device may result in improper operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.24.4 Including the SPI Framework Module in an Application

This section describes how to include the SPI Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SPI Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the SPI Framework module is g_sf_spi_device0. This name can be changed in the associated Properties window.)

SPI Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_spi_device0 on sf_spi	Threads	**New Stack> Framework> Connectivity> SPI Framework Device on sf_spi**

When the SPI Framework module on sf_spi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

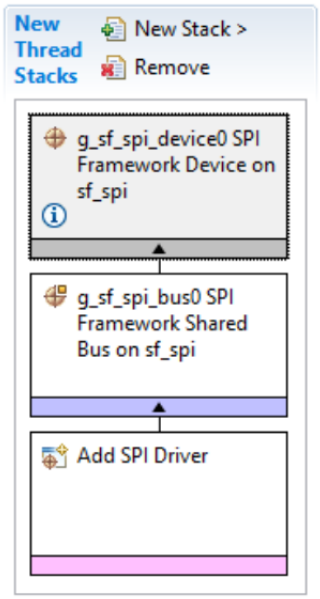


Figure 202: SPI Framework Module Stack

4.1.24.5 Configuring the SPI Framework Module

The SPI Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the SPI Framework Device Module on sf_ spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_spi_device0	Module name.

Clock Phase	Data sampling on odd edge, data variation on even edge/Data sampling on even edge, data variation on odd edge Default: Data sampling on odd edge, data variation on even edge	Select the clock phase.
Clock Polarity	Low when idle, High when idle Default: Low when idle	Select the clock polarity.
Chip Select Port	00 thru 11 Default: 00	Select GPIO port used for the chip select.
Chip Select Pin	00 thru 15 Default: 00	Select GPIO pin used for the chip select.
Chip Select Active Level	Low, High Default: Low	Polarity of the Chip Select signal, active High or Low.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different chip-select GPIOs or levels. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the SPI Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules:

Configuration Settings for the SPI Framework Shared Bus on sf_spi

ISDE Property	Value	Description
Name	g_sf_spi_bus0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the RSPI HAL Driver on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.
Clock Polarity	Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first.
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	SPI Operation, Clock synchronous operation Default: SPI Operation	Select spi or clock syn mode operation.
Slave Select Polarity(SSL0)	Active Low, Active High Default: Active Low	Select SSL0 signal polarity.
Slave Select Polarity(SSL1)	Active Low, Active High Default: Active Low	Select SSL1 signal polarity.
Slave Select Polarity(SSL2)	Active Low, Active High Default: Active Low	Select SSL2 signal polarity.
Slave Select Polarity(SSL3)	Active Low, Active High Default: Active Low	Select SSL3 signal polarity.

Select Loopback1	Normal, Inverted Default: Normal	Select the data mode for loopback 1.
Select Loopback2	Normal, Inverted Default: Normal	Select the data mode for loopback 2.
Enable MOSI Idle State	Enable, Disable Default: Disable	Select MOSI idle fixed value and selection.
MOSI Ildle State	MOSI Low, MOSI High Default: MOSI Low	Select mosi idle fixed value and selection.
Enable Parity	Enable, Disable Default: Disable	Enable/disable parity.
Parity Mode	Parity Odd, Parity Even Default: Parity Odd	Select parity.
Select SSL(Slave Select)	SSL0, SSL1, SSL2, SSL3 Default: SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3.
Select SSL Level After Transfer	SSL Level Keep, SSL Level Do Not Keep Default: SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep.
Clock Delay Enable	Clock Delay Enable, Clock Delay Disable Default: Clock Delay Disable	Clock delay enable selection.
Clock Delay Count	Clock Delay 1 thru 8 RSPCK Default: Clock Delay 1 RSPCK	Clock delay count selection.
SSL Negation Delay Enable	Negation Delay Enable, Negation Delay Disable Default: Negation Delay Disable	SSL negation delay enable selection.
Negation Delay Count	Negation Delay 1 thru 8 RSPCK Default: Negation Delay 1 RSPCK	Negation delay count selection.
Next Access Delay Enable	Next Access Delay Enable, Next Access Delay Disable Default: Next Access Delay Disable	Next access delay enable selection.

Next Access Delay Count	Next Access Delay 1 thru 8 RSPCK Default: Next Access Delay 1 RSPCK	Next access delay count selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.

Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.

Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI Driver on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.
Clock Polarity	Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first.

Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable.
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Bitrate Modulation Function enable or disable. Note: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.

Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.

Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SPI Framework Module Clock Configuration

The SCI peripheral module uses the PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator **Clock** tab prior to a build or by using the CGC Interface at run-time.

SPI Framework Module Pin Configuration

The SPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note

For some peripherals, the operation-mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for the SPI Framework Module

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals**> **Connectivity: SCI**> **SCI1**

RSPI	Pins	Select **Peripherals**> **Connectivity: SPI**> **SPI0**

Note

The selection sequences assume SCI1 and SPI0 are the desired hardware targets of the drivers.

Pin Configuration Settings for the SPI Framework Module

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Simple SPI as the Operation Mode for SPI on SCI.
CTS0_RTS0_SS0	None, P103, P413 Default: None	SS0 pin selection.
RXD0_SCL0_MISO0	None, P100, P410 Default: None	MISO0 pin selection.
SCK0	None, P102, P412 Default: None	SCK0 pin selection.
TXD1_SDA1_MOSI0	None, P213, P709 Default: None	MOSI0 pin selection.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

SPI Framework Module Additional Settings

If external chip selects are being used, configure the chip select pins as GPIO outputs.

4.1.24.6 Using the SPI Framework Module in an Application

A common application for the SPI framework module requires multiple slave devices on a single bus. The implementation for this common application is described below. A second implementation shows two buses each with two slave devices attached.

Implementation Steps for Two Slave Devices on a Single Shared Bus

When using the SPI framework module to create a single bus with multiple slave devices create two thread stacks each with an SPI framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note

The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.

Step 1: Add the first SPI framework device module to a new or existing thread. This creates the SPI master stack. A shared bus on sf_spi is added along with the SPI driver. The SPI driver can be selected for implementation on r_rspi or r_sci_spi. The DTC transfer driver is also added by default. This can be removed if the CPU transfer mode is needed instead.

The resulting module stack is shown in the following figure. Example configuration settings are given in the tables that follow the figure.

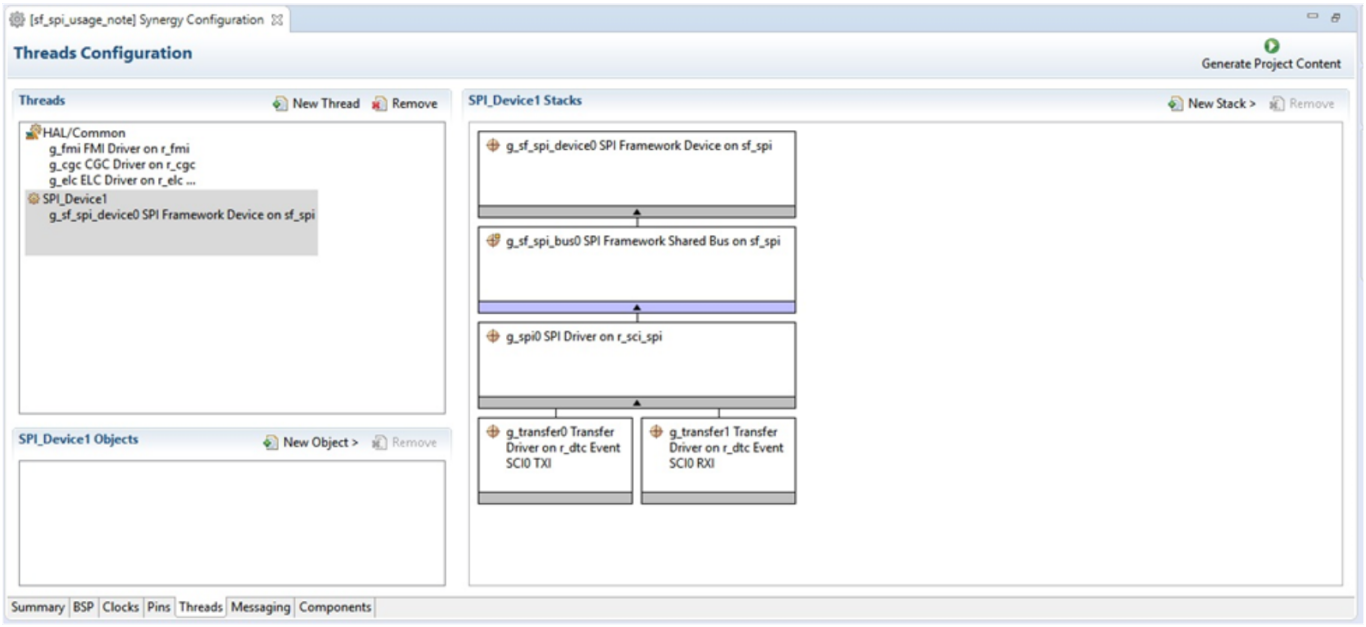


Figure 203: SPI Framework Module Stack 1

Example configuration settings for the key first thread stack modules for Slave Device #1 are as follows:

Configuration Settings for the SPI Framework Module on sf_spi

Property	Value	Description
Parameter Checking	Disabled	Enable or Disable Parameter Checking.
Name	g_sf_spi_device1	Give a name to identify the SPI Framework device. API, Config and Control instances will be created based on this name.
Clock Phase	Data sampling on odd edge	Specify the clock phase for data variation and data sampling
Clock Polarity	Low when idle	Select the clock polarity when clock is idle.
Clock Select Port	01	Select GPIO port used for the chip select.
Chip Select Pin	04	Select GPIO pin used for the chip select.

Chip Select Active Level	Low	Select Polarity of the chip select signal.

Configuration Settings for the SPI Framework Shared Bus on sf_spi

Property	Value	Description
Name	g_sf_spi_bus0	Give a name to identify the SPI Framework shared bus. This shared bus will be shared by multiple SPI Framework Devices.

Configuration Settings for the SPI Driver on r_rspi

Property	Value	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_spi0	Give a name to identify the SPI Driver device. This will be used by Framework internally.
Channel	0	Channel number.
Operating Mode	Master	Operating mode selection.
Clock Phase	Data sampling on odd edge/ data variation on edge	Clock phase selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Clock Polarity	Low when idle	Clock polarity selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Mode Fault Error	Enable	Mode fault error selection.
Bit Order	MSB First	Bit order selection.
Bitrate	500000	Bit rate selection.
Callback	NULL	Callback function name.
SPI Mode	SPI Operation	SPI mode selection.
SPI Communication Mode	Full Duplex	SPI communication mode selection.
Slave Select Polarity (SSL0)	Active Low	Slave select polarity selection 0.
Slave Select Polarity (SSL1)	Active Low	Slave select polarity selection 1.

Slave Select Polarity (SSL2)	Active Low	Slave select polarity selection 2.
Slave Select Polarity (SSL3)	Active Low	Slave select polarity selection 3.
Select Loopback 1	Normal	Loopback 1 selection.
Select Loopback 2	Normal	Loopback 2 selection.
Enable MOSI Idle	Disable	Enable MOSI idle selection.
MOSI Idle State	MOSI Low	Enable MOSI idle state selection.
Enable Parity	Disable	Enable parity selection.
Parity Mode	Parity Odd	Enable parity mode selection.
Select SSL (Slave Select)	SSL0	Select SSL selection.
Select SSL Level After Transfer	SSL Level Keep	Select SSL level after transfer selection.
Clock Delay Enable	Disable	Clock delay enable selection.
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection.
SSL Negation Delay Enable	Disable	SSL Negation Delay Enable selection.
Negation Delay Count	Clock Delay 1 RSPCK	Negation Delay Count selection.
Next Access Delay Enable	Disable	Next Access Delay Enable selection.
Next Access Delay Count	Clock Delay 1 RSPCK	Next Access Delay Count selection.
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.

Configuration Settings for the SPI Driver on r_sci_spi

Property	Value	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_spi0	Give a name to identify the SPI Driver device. This will be used by Framework internally.
Channel	0	Channel number.
Operating Mode	Master	Operating mode selection.

Clock Phase	Data sampling on odd edge/ data variation on even edge	Clock phase selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Clock Polarity	Standard	Clock polarity selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Mode Fault Error	Disable	Mode fault error selection.
Bit Order	MSB First	Bit order selection.
Bitrate	500000	Bit rate selection.
Bit Rate Modulation Enable	Enable	Enables/Disable the bit rate modulation.
Callback	NULL	Callback function name. This field will be locked as callback is handled internally in the framework.
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 2	Error interrupt priority selection.

Note

DTC configuration settings are not shown as a simplification.

Step 2: Add the second SPI Framework Device to a different thread. The SPI Framework Shared Bus on sf_spi is not added automatically. To add it, select the option to use the existing shared bus. The configurator will then automatically add the SPI Framework Shared Bus on sf_spi and the remaining modules. The lower level modules will automatically be configured to be consistent with the previously defined settings from the first SPI framework instance. This ensures that the SPI driver configurations are the same for both devices except for the Clock Phase, Clock Polarity, Chip Select Pin and Port, and Chip Select Active Level properties, as these are defined under the SPI Framework Device module and can be different for each slave device.

The resulting module stack is shown in the following figure:

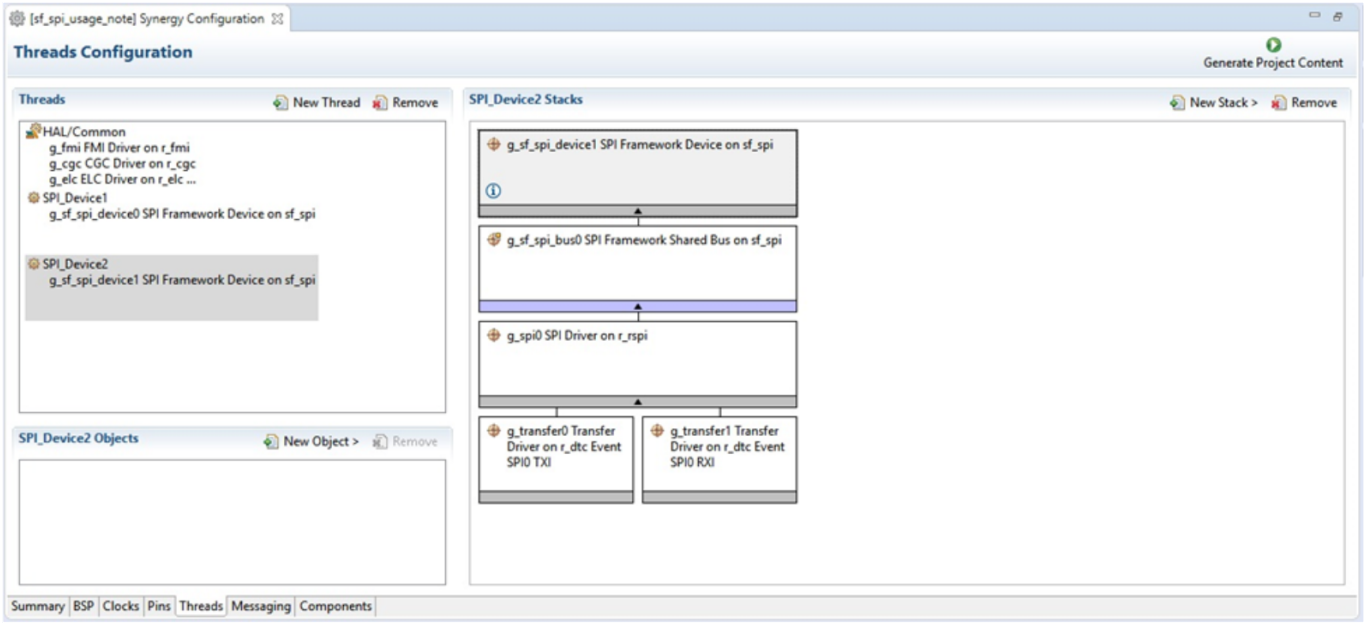


Figure 204: SPI Framework Module Stack 2

The only differences in the configuration parameters for the second stack are the name for the second SPI framework device module, and any differences in the non-shared slave settings (Clock Phase, Clock Polarity, Clock Select Port, Chip Select Pin and Chip Select Active Level). Example settings are shown in the following table:

Configuration Settings for the SPI Framework Device on sf_spi (Slave #2)

Property	Value	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_sf_spi_device2	Give a name to identify the SPI Framework device. API, Config and Control instances will be created based on this name.
Clock Phase	Data sampling on odd edge/ data variation on even edge	Specify the clock phase for data variation and data sampling
Clock Polarity	High when idle	Select the clock polarity when clock is idle.
Clock Select Port	05	Select GPIO port used for the chip select.
Chip Select Pin	01	Select GPIO pin used for the chip select.
Chip Select Active Level	Low	Select Polarity of the chip select signal.

Implementation Steps for Two Slave Devices on Two Shared Buses

When using the SPI framework module to create a single bus with multiple slave devices create two thread stacks each with an SPI framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note

The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.

Adding Another Shared Bus

To add another shared bus, just follow the below steps. The previous example is used as the starting point.

Step 3: The SPI framework module which will use a second shared bus can be added to any thread. Starting with the previous example, if it is added to the SPI_Device1 thread, then the module stack would appear as shown below. Available options for the shared bus are New or Use.

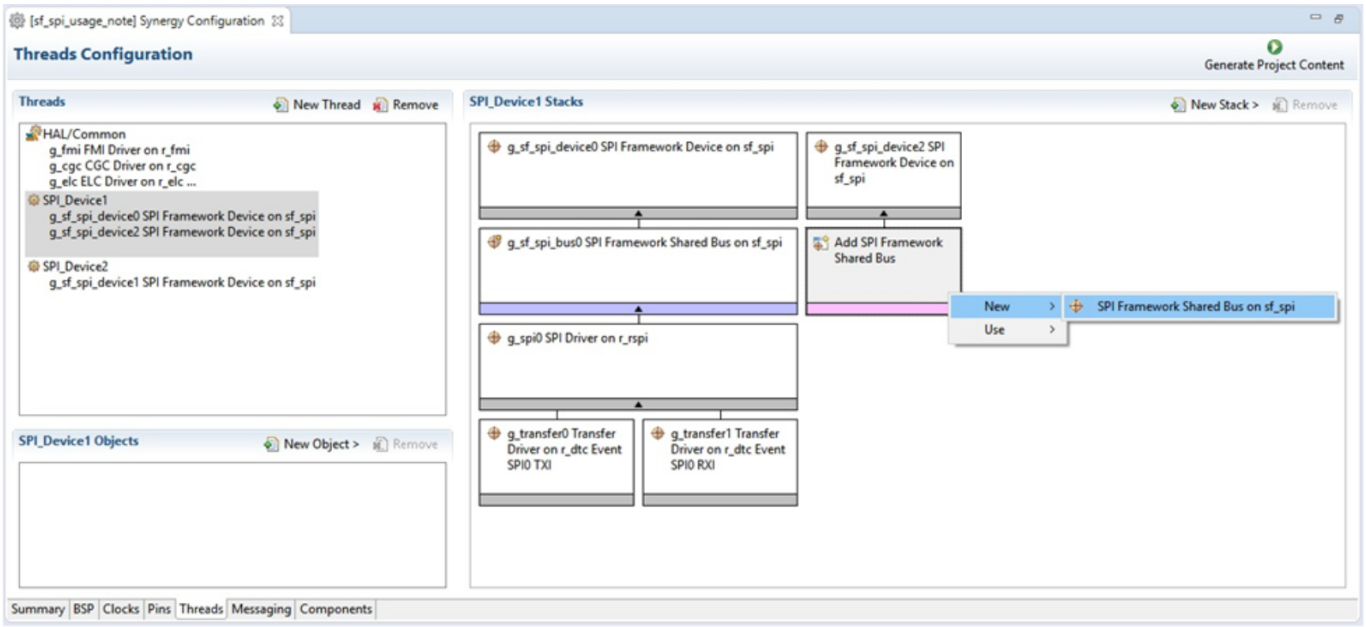


Figure 205: SPI Framework Module Stack 3

Step 4: Select New to and add another SPI Framework Shared Bus on sf_spi module. Configure the shared bus properties as needed for the application. Select the desired low-level SPI driver. The channel number for the g_spi1 SPI driver module, must be different from the channel number for the g_spi0 SPI driver module. The resulting thread stack is shown below:

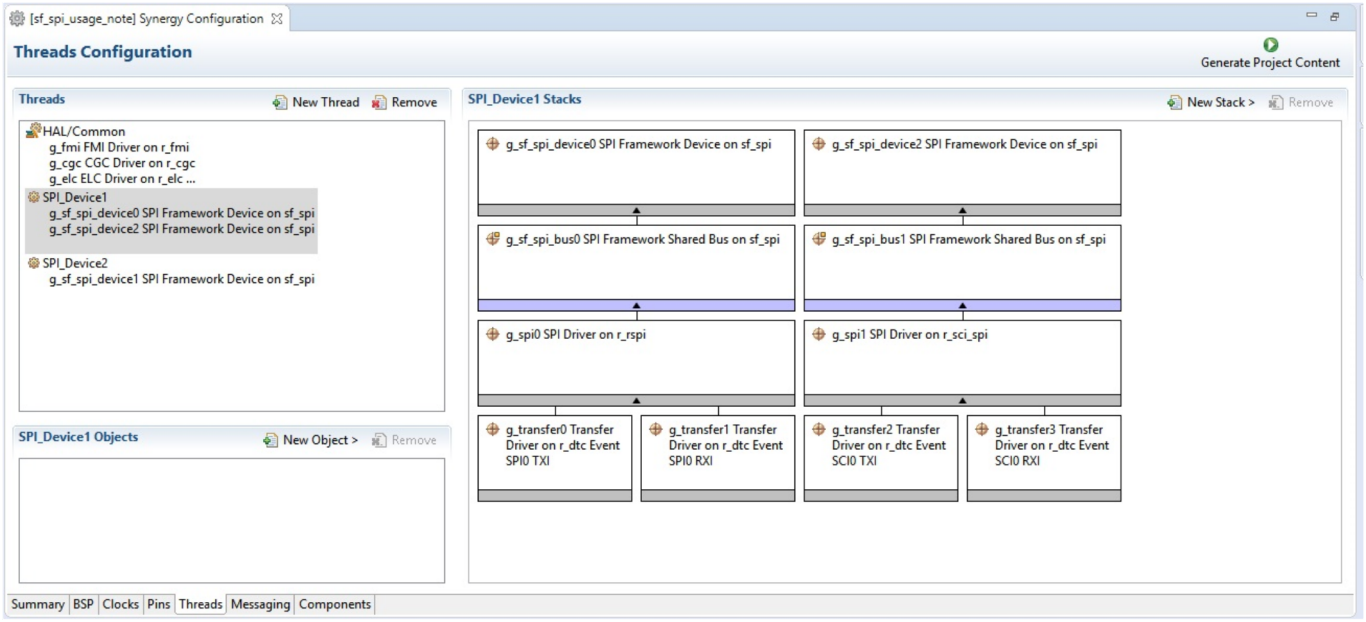


Figure 206: SPI Framework Module Stack 4

Step 5: A second device can be added in the SPI_Device2 thread using the same steps described above. The resulting thread stack is shown below:

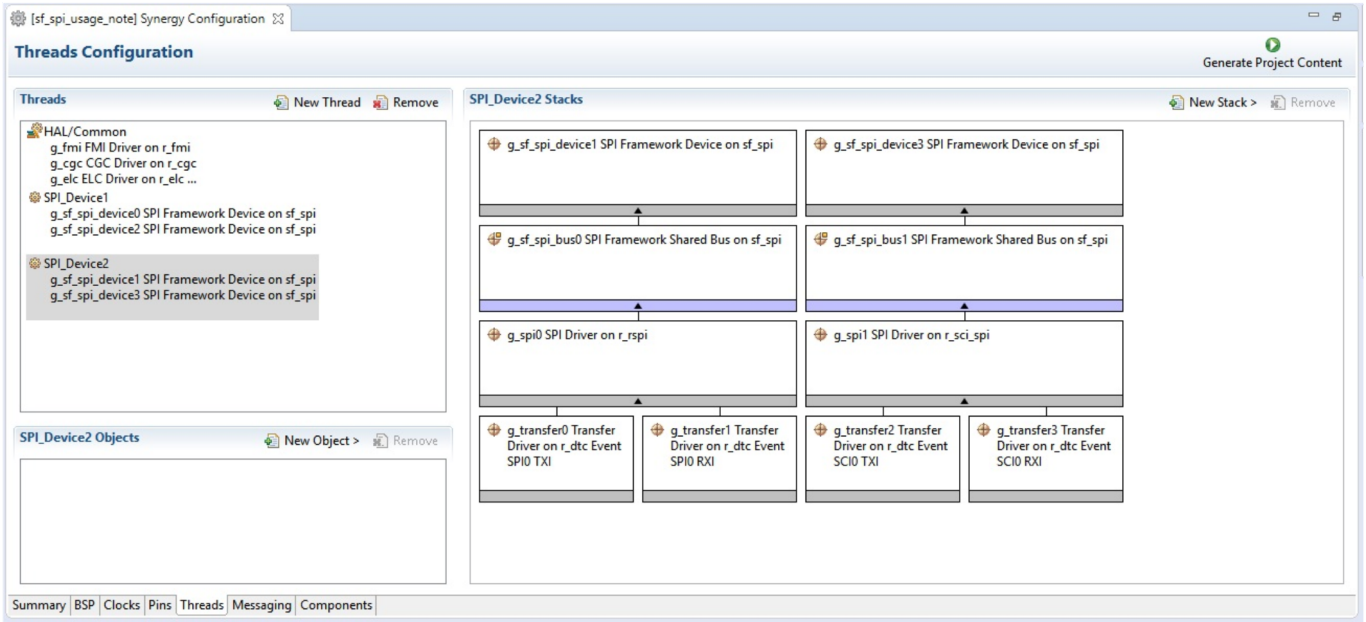


Figure 207: SPI Framework Module Stack 5

The typical steps in using the SPI Framework module in an application are:

1. Initialize the SPI Framework device module using the sf_spi_api_t::open API function. Each SPI framework device module needs to call the spi_api_t::open API function at least once before performing any operations on the bus.
2. Lock the bus for continuous transfer using the sf_spi_api_t::lock or sf_spi_api_t::lockWait API

function for a particular SPI Framework device module. Once the bus is locked by a particular SPI Framework device module, it cannot be used by any other SPI Framework device module on the bus. This ensures that ownership of the bus remains with the locked module until it explicitly unlocks it. Any kind of operation from other SPI Framework device modules on the bus will return a fail status during this period. It is not mandatory to lock the bus before any read/write operations on the bus. It is optional.

3. Read data using the sf_spi_api_t::read API function. The read operation will not be successful if the bus is already locked by any other SPI Framework device module.
4. Write data using the sf_spi_api_t::write API function. The write operation will not be successful if the bus is already locked by any other SPI Framework device module.
5. Write and read data simultaneously using the sf_spi_api_t::writeRead API function. The simultaneous read and write operation will not be successful if the bus is already locked by any other SPI Framework device module.
6. Unlock the bus from continuous transfer using the sf_spi_api_t::unlock API function if it is already locked by the same device. Once the bus is unlocked, other SPI Framework device modules can use it. It is necessary to unlock the locked bus after the intended read/write operation is completed.
7. Close the SPI Framework device module using the sf_spi_api_t::close API function. Each SPI Framework device module can call the sf_spi_api_t::close API function after all read/write operations on the bus are over.

These common steps are illustrated in a typical operational flow in the following figure:

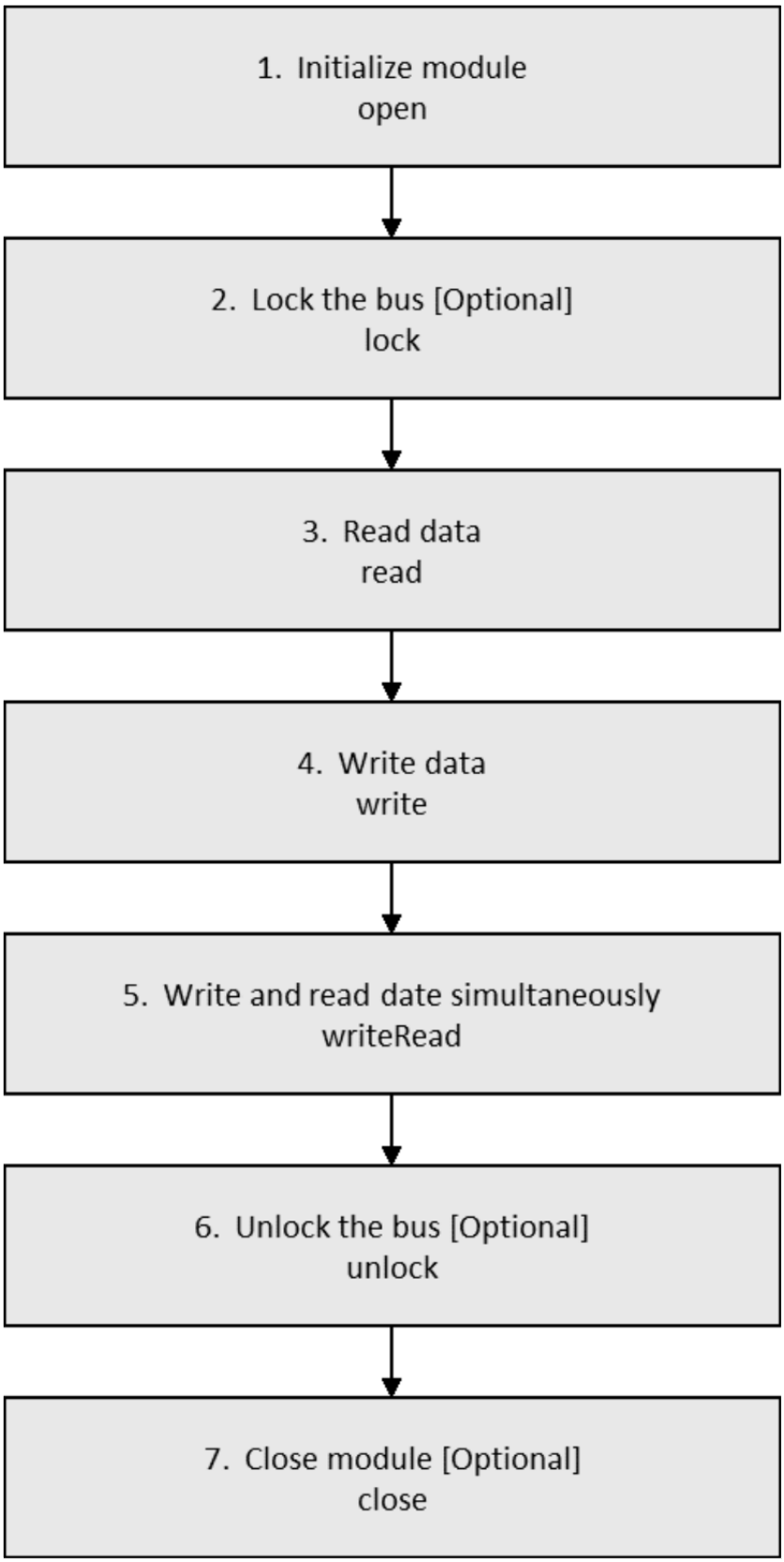


Figure 208: Flow Diagram of a Typical SPI Framework Module Application

4.1.25 Thread Monitor Framework

4.1.25.1 Thread Monitor Framework Module Introduction

The Thread Monitor Framework provides a high-level API for system monitoring applications using the watchdog timer (WDT) or independent watchdog timer (IWDT) to monitor program execution. The Thread Monitor Framework uses the WDT or IWDT peripherals on the Synergy MCU device.

Thread Monitor Framework Module Features

- The Thread Monitor Framework interface monitors RTOS threads using a watchdog timer. The Thread Monitor forces a watchdog reset of the microcontroller when any of the monitored threads do not behave as expected.
- The Thread Monitor is designed to support any Synergy device with either a WDT or IWDT peripheral and a HAL module with no changes to the API.
- In profiling mode, the minimum and maximum counter values for registered threads can be determined. When in profiling mode, the watchdog timer is always refreshed and does not reset the device.
- Both the WDT and IWDT HAL modules are supported by this framework module.

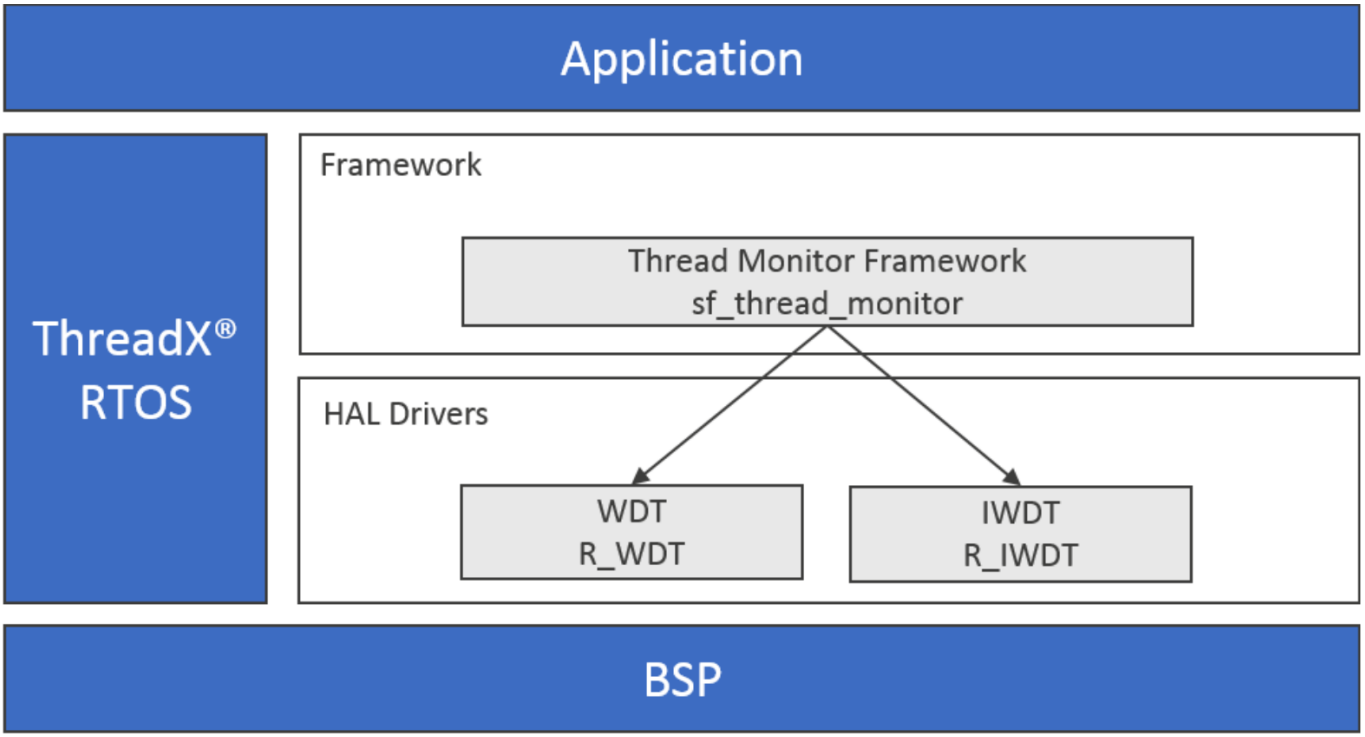


Figure 209: Thread Monitor Framework Module Block Diagram

4.1.25.2 Thread Monitor Framework Module APIs Overview

The Thread Monitor Framework defines APIs for opening and closing the framework and registering and unregistering threads for monitoring. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of return status values follows the API summary table.

Thread Monitor Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_thread_monitor.p_api->open (g_sf_thread_monitor.p_ctrl,g_sf_thread_monitor.p_cfg); Configures the WDT or IWDT module. From the configuration data, the timeout period of the WDT/IWDT is determined. A thread created to monitor registered threads.
close	g_sf_thread_monitor.p_api->close (g_sf_thread_monitor.p_ctrl); Suspends the thread monitoring thread. The watchdog peripheral no longer refreshes.
threadRegister	g_sf_thread_monitor.p_api-> threadRegister (g_sf_thread_monitor.p_ctrl, &p_min_max_struct); Registers a thread for monitoring.
threadUnregister	g_sf_thread_monitor.p_api-> threadUnregister (g_sf_thread_monitor.p_ctrl); Removes a thread from monitoring.
countIncrement	g_sf_thread_monitor.p_api-> countIncrement (g_sf_thread_monitor.p_ctrl); Safely increments a monitored thread's count value.
versionGet	g_sf_thread_monitor.p_api-> versionGet(&version); Retrieves the API version and stores it in the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Pointer is null.
SSP_ERR_IN_USE	Thread monitor has already been opened.
SSP_ERR_INVALID_MODE	Low-level watchdog peripheral returns an error when opened.
SSP_ERR_UNSUPPORTED	Data structure could not be allocated.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid for the low-level driver.

SSP_ERR_NOT_OPEN	sf_thread_monitor_api_t::open has either not been called or was not called successfully.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads-to-be-monitored array to add this thread. Increases the value of THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS in sf_thread_moinitor_cfg.h.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.25.3 Thread Monitor Framework Module Operational Overview

The Thread Monitor performs as follows: a thread registers a counter variable with the thread monitor along with the minimum and maximum expected values for this counter variable. The thread which is monitored increments the counter variable while it runs. At a period of half the watchdog timeout period, the thread monitor checks the counter variables of registered threads. If any fall outside of the minimum and maximum values, the watchdog timer is allowed to reset the microcontroller. If all fall within their expected range, the watchdog timer is refreshed and the counter variables are cleared to zero.

Thread Monitor Framework Module Important Operational Notes and Limitations

Thread Monitor Framework Module Operational Notes

The following figure shows a flowchart for the operation of the Thread Monitor Framework module.

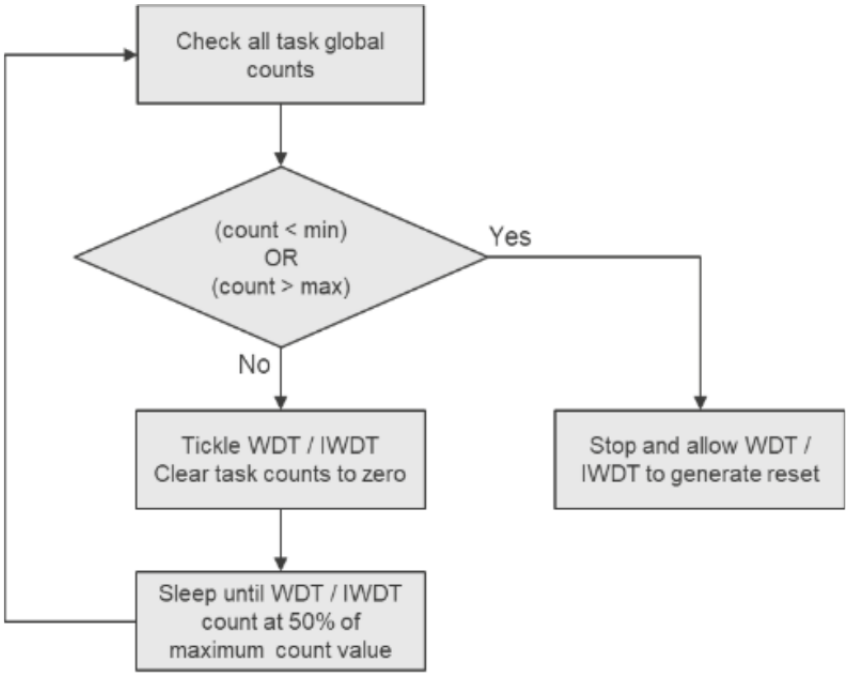


Figure 210: Thread Monitor Framework Operation Diagram

The following figure shows when the WDT/IWDT refreshes. Note that the valid refresh period is the central 50% of the count period, 25% on either side of the 50% count value.

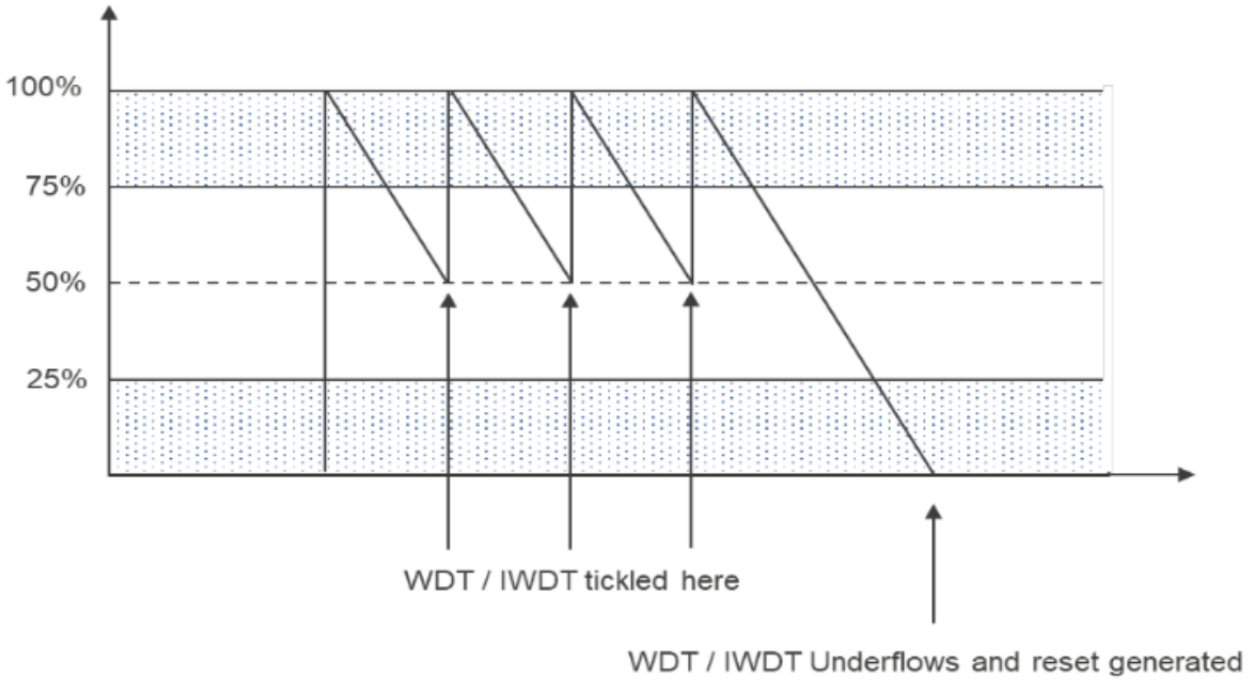


Figure 211: WDT/IWDT Refresh Operation

- The IWDT has its own clock source to improve safety.
- The WDT can be started from the application.
- When the thread is not executing sleep mode, set the stop-control property of the WDT to WDT Count Enabled in Low Power Mode. When the thread is not executing (asleep), the ThreadX® executes a WFI instruction effectively putting the device into soft sleep, which causes the WDT to stop counting.

Note

Do not open or refresh the WDT in a monitored thread file; it is done automatically by the Thread Monitor Framework.

Internally, the Thread Monitor Framework runs at the rate of half of the watchdog timer reset period. This rate ensures the Thread Monitor Framework runs at 50% of the watchdog count value-well within the valid refresh window. The Thread Monitor calculates the reset period internally by querying the lower-level watchdog driver.

Thread Monitor Framework Profiling Calculation

The thread monitor framework provides a method to keep track of the number of times a thread is registered. Any of the threads registered with the thread monitor framework need to call the sf_thread_monitor_api_t::countIncrement API to update a counter which is an indication of the number of times a specified thread was executed. A thread, which is part of the thread monitor framework would run at a predefined time interval and keeps the count value of the thread registered for monitoring. The monitor task will analyze count values taken over each interval and keeps track of the minimum and maximum counts obtained for each of the threads.

For example, consider a WDT configured with a time-out cycle of 16384 and clock division ratio of 8192. Assuming a PCLK of 60 MHz and time for a system tick to be 10 ms, the total number of ticks for WDT time out would be 223. Considering 50% of WDT time out, the thread monitor task would run at every 111 ticks to keep track of the threads registered for monitoring.

In a scenario where a simple thread executing a continuous loop with a sleep of 2 ticks in between each iteration, the max count value would be 56 (that is, 111/2).

When the thread monitor is run for the first time, it waits a full WDT time-out period of 111 counts (223 ticks) before activating the framework. This feature ensures proper working of WDT. In addition, whenever a new thread is registered, the max/min count value will not update unless the registered thread is activated from the thread monitor. This results in an additional delay of 56 counts (111 ticks).

Note

For the very first task registered with the thread monitor, a total delay of 111 + 56 counts would be introduced after which the Min/Max count is updated. The profiling calculation remains same for both WDT and IWDT clock sources except for the refresh time-out period.

Thread Monitor Framework Module Limitations

- The Thread Monitor Framework has a sf_thread_monitor_api_t::close API call. When WDT and IWDT are being used, it is not possible to stop them. If the Thread Monitor Framework is closed, some other provision for refreshing the watchdog must be made or the device resets.
- The Thread Monitor Framework has a sf_thread_monitor_api_t::threadUnregister API call which removes the calling thread from being monitored. The API would return SSP_SUCCESS for any thread that is not registered with the sf_thread_monitor framework and also for threads which were already unregistered.
- Debugging mode-support is required when running with a J-Link on some devices; WDT/IWDT does not count when using J-Link debugging hardware. The Thread Monitor Framework thread typically synchronizes to the WDT/IWDT counter, but skips this synchronization step when it is running with J-Link.
- Assign a high priority (low number in ThreadX) to the Thread Monitor Framework thread; any delays in running the Thread Monitor Framework could refresh the watchdog outside of the valid refresh window, causing the microcontroller to reset. The Thread Monitor Framework thread does not run for long and does not impact the performance of the system.
- Refer to the latest *SSP Release Notes* for any additional operational limitations for this module.

4.1.25.4 Including the Thread Monitor Framework Module in an Application

This section describes how to include the Thread Monitor Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Thread Monitor Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Thread Monitor Framework is g_sf_thread_monitor. This name can be changed in the associated Properties window.)

Thread Monitor Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_thread_monitor0 Thread Monitor Framework	Threads	**New Stack> Framework> Services> Thread Monitor Framework on sf_thread_monitor**

When the Thread Monitor Framework on sf_thread_monitor is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

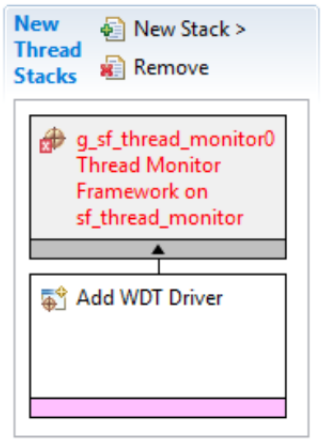


Figure 212: Thread Monitor Framework Module Stack

4.1.25.5 Configuring the Thread Monitor Framework Module

The Thread Monitor Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Thread Monitor Framework Module on sf_thread_monitor

ISDE Property	Value	Description

Parameter Checking	Enabled, Disabled, BSP Default: BSP	Controls whether to include code for API parameter checking.
Maximum Number of Monitored Threads	5	Maximum number of threads that can be monitored.
Name	g_sf_thread_monitor0	The name of the Thread Monitor instance.
Profiling Mode	Enabled, Disabled, Default: Disabled	Whether profiling mode should be enabled.
Thread Monitor Thread Priority	1	Priority of thread monitor internal thread.
Name of generated initialization function	sf_thread_monitor_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Thread Monitor Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the Independent Watchdog Timer on r_iwdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_wdt0	Module name.

NMI Callback	NULL	Callback. A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Watchdog Timer on r_wdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_wdt0	Module Name.
Start Mode	Register, Auto Default: Register	Configures the start mode as register start or auto-start.
Start Watchdog After Configuration	True, False Default: True	Controls whether WDT is started during initialization.
Timeout	1024 cycles, 4096 cycles, 8192 cycles, 16384 cycles Default: 16384 cycles	WDT timeout period.
Clock Division Ratio	PCLK/4, PCLK/64, PCLK/128, PCLK/512, PCLK/2048, PCLK/8192 Default: PCLK/8192	WDT clock divider.
Window Start Position	100% (Window Position Not Specified), 75%, 50%, 25% Default: 100% (Window Position Not Specified)	Permitted refresh period start postion.

Window End Position	0% (Window Position Not Specified), 25%, 50%, 75% Default: 0% (Window Position Not Specified)	Permitted refresh period end postion.
Reset Control	Reset Outpout, NMI Generated Default: Reset Output	Select whether WDT should reset the MCU or generate an NMI.
Stop Control	WDT Count Enabled in Low Power Mode, WDT Count Disabled in Low Power Mode Default: WDT Count Disabled in Low Power Mode	Select whether the WDT should stop counting in low power modes.
NMI Callback	NULL	Callback. A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Thread Monitor Framework Module Clock Configuration

Use the ISDE to configure the WDT clock using the **Clocks** tab.

The WDT is initially based on the PCLKB frequency. Set the PCLKB frequency in the ISDE using the clock configurator or the CGC Interface at run-time.

With the PCLKB running at 60 MHz, the WDT maximum timeout period is approximately 2.24 seconds.

The IWDT clock runs at 15 kHz resulting in a maximum possible timeout period of just under 35 seconds.

Thread Monitor Framework Module Pin Configuration

The Thread Monitor Framework does not require any pins for its operation.

Threads for the Thread Monitor Framework Module Application

Any thread monitored by the Thread Monitor Framework must be instrumented to work with the monitoring module. When a thread to be monitored is registered with the Thread Monitor, the expected minimum and maximum count values are passed via a pointer to a structure of type sf_thread_monitor_counter_min_max_tcontaining the values.

The thread's counter and minimum and maximum values must be registered with the Thread Monitor Framework by calling g_sf_thread_monitor.p_api->threadRegister().

Each time round the monitored thread's loop, the counter value should be updated by calling g_sf_thread_monitor.p_api->countIncrement().

Other Thread Monitor Framework Module Settings

The Thread Monitor Framework does not use any interrupts; the WDT/IWDT must be configured to generate a reset.

4.1.25.6 Using the Thread Monitor Framework Module in an Application

The steps in using the Thread Monitor Framework module on sf_thread_monitor in a typical application are:

1. Initialize the Thread Monitor using the sf_thread_monitor_api_t::open API.
2. Register thread that needs to be monitored with the sf_thread_monitor_api_t::threadRegister API.
3. Use the sf_thread_monitor_api_t::countIncrement API to increment the count every time the registered thread is executed.
4. Unregister the thread with the sf_thread_monitor_api_t::threadUnregister API when there is no longer a need to monitor the registered thread.
5. Close the Thread Monitor with the sf_thread_monitor_api_t::close API (only if the Thread Monitor is no longer required in an application).

These common steps are illustrated in a typical operational flow diagram in the following figure:

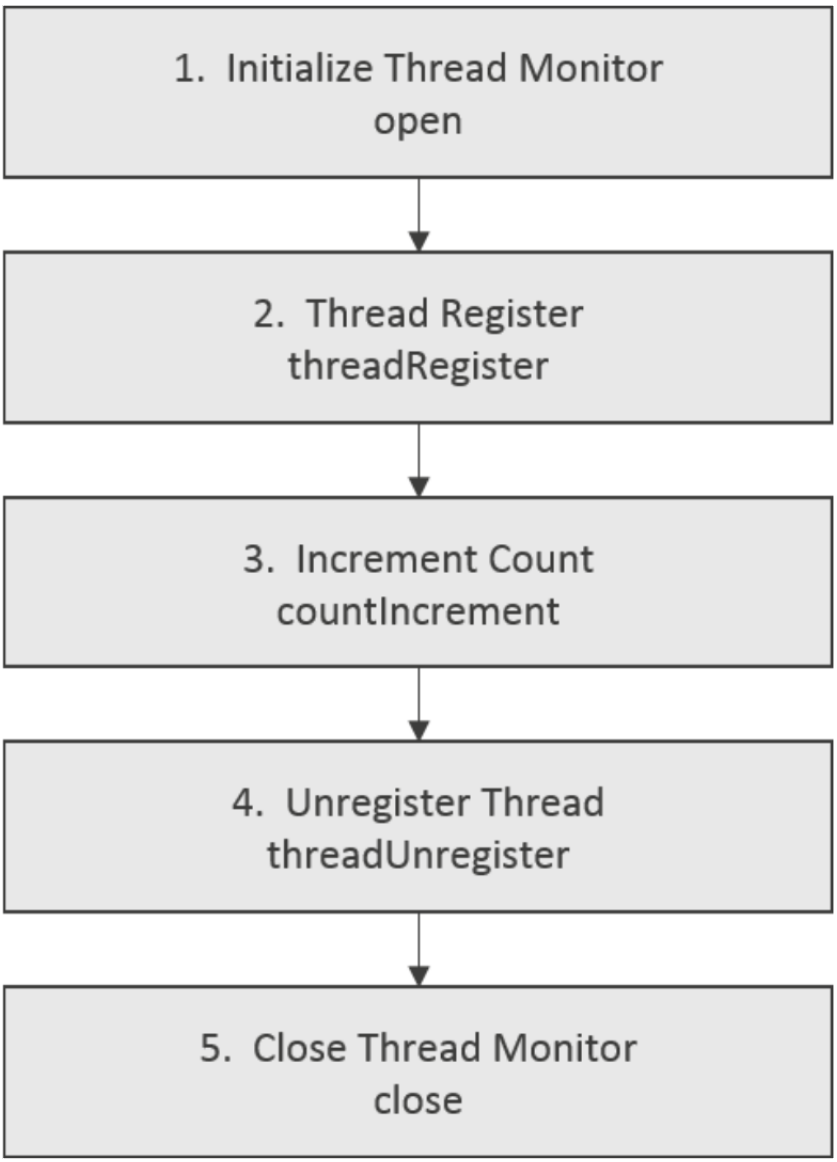


Figure 213: Flow Diagram of a Typical Thread Monitor Framework Module Application

4.1.26 Touch Panel V2 Framework

4.1.26.1 Touch Panel V2 Framework Introduction

The Touch Panel V2 Framework module provides a high-level API for obtaining touch data on coordinates and events from the touch controller. The Touch Panel V2 Framework module uses the touch chip driver SSP Supplement module for communication with the touch panel.

Touch Panel V2 Framework Module Features

- Provides position data (X and Y coordinates).
- Provides rotation of touch coordinates.

- Provides the touch event type (down, up, move, hold, or invalid).
- A callback can be registered or an API function can be used to get touch data.
- Supports calibration of the touch panel for key elements: scalar, rotation and mechanical shifts.
- Provides a common API interface to touch chip drivers.
- Supports an adjustable update frequency.

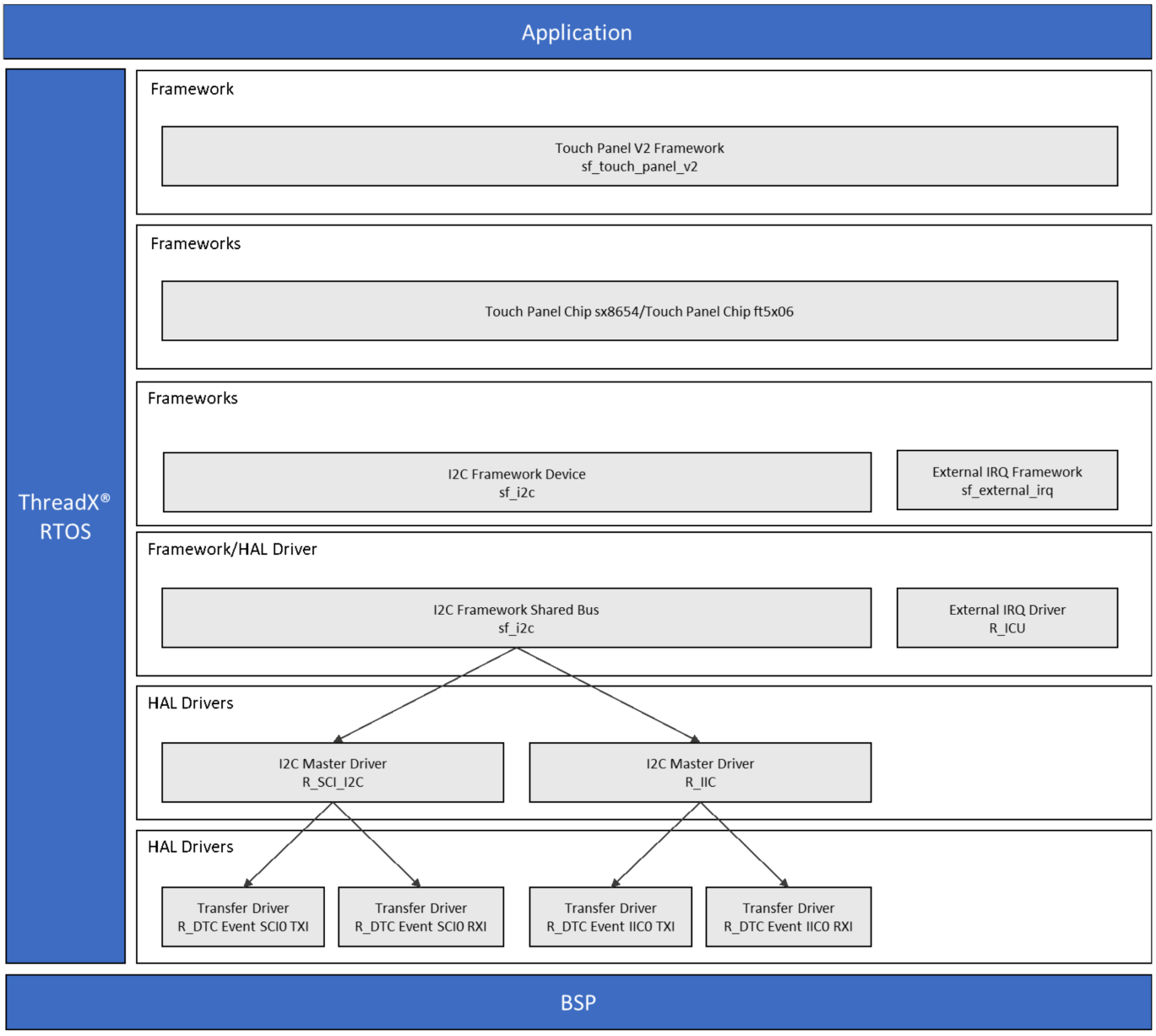


Figure 214: Touch Panel V2 Framework Module Block Diagram

4.1.26.2 Touch Panel V2 Framework Module APIs Overview

The Touch Panel V2 Framework module defines API functions such as opening, calibrating, starting, stopping or closing. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Touch Panel V2 Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_touch_panel_v2_0.p_api->open(g_sf_touch_panel_v2_0.p_ctrl, g_sf_touch_panel_v2_0.p_cfg); Create required RTOS objects, call a lower level module for hardware specific initialization and create a thread to post touch data to a user application.
calibrate	g_sf_touch_panel_v2_0.p_api->calibrate(g_sf_touch_panel_v2_0.p_ctrl, p_display, p_touchscreen, timeout); Begin calibration routine based on provided expected and actual coordinates.
start	g_sf_touch_panel_v2_0.p_api->start(g_sf_touch_panel_v2_0.p_ctrl); Start scanning for touch events.
stop	g_sf_touch_panel_v2_0.p_api->stop (g_sf_touch_panel_v2_0.p_ctrl); Stop scanning for touch events.
touchDataGet	g_sf_touch_panel_v2_0.p_api->touchDataGet (g_sf_touch_panel_v2_0.p_ctrl, p_payload, timeout); Reads the touch data and fills in the touch payload data.
reset	g_sf_touch_panel_v2_0.p_api->reset(g_sf_touch_panel_v2_0.p_ctrl); Resets touch controller if reset pin is provided.
close	g_sf_touch_panel_v2_0.p_api->close(g_sf_touch_panel_v2_0.p_ctrl); Terminates touch thread and closes channel at HAL layer.
versionGet	g_sf_touch_panel_v2_0.p_api->versionGet(g_sf_touch_panel_v2_0.p_version); Retrieves API version and stores it in the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	A pointer parameter was NULL, or a lower level driver reported this error.

SSP_ERR_INTERNAL	The touch panel thread or event flags could not be created, or a lower level driver reported this error.
SSP_ERR_CALIBRATE_FAILED	Actual touch value was not in expected range.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.
SSP_ERR_IN_USE	Mutex was not available, or a lower level driver reported this error.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.26.3 Touch Panel V2 Framework Module Operational Overview

The Touch Panel V2 Framework module scans data from a touch controller and invokes the user registered callback. If user callback is not registered the sf_touch_panel_v2_api_t::touchDataGet API function can be used to retrieve the data. The touch data contains touch coordinates (X and Y coordinates) and the touch event type (down, up, move, hold, or invalid). The Touch Panel V2 Framework uses an internal thread to read the touch controller.

Touch Panel V2 Framework Module Important Operational Notes and Limitations

Touch Panel V2 Framework Module Operational Notes

The key operational elements of the Touch Panel V2 Framework are described below:

Auto Initialization and Auto Start

The "Auto Initialization" property in the Synergy configurator window must be enabled to automatically initialize the framework. The "Auto start" property in the configurator window must be enabled to automatically receive touch data from the touch chip.

User Callback

The user callback must be registered to obtain touch data. If the user callback is registered then the framework will invoke the callback when touch events occur.

API to Get the Touch Data

The sf_touch_panel_v2_api_t::touchDataGet API function can be used to obtain touch data. This function will wait until new touch event data is available or it will timeout (based on the timeout argument passed to the API).

Update Frequency

The application will be notified with repeated touch events (like touch event down and touch event hold and touch event invalid) at the specified update frequency (Update Hz) configured in the Synergy configurator property window.

Note

The touch event UP and touch event DOWN will be notified regardless of the update frequency.

For example,

If the "Update Hz" property is set to 10 Hz, then only 10 repeated touch events will be notified to the application in a second.

Calibration

The touch panel V2 framework supports calibration of touch data to overcome the issues in the touch panel like: Scalar, rotation and Mechanical shifts.

To obtain the calibrated data the user needs to call the sf_touch_panel_v2_api_t::calibrate API function and pass 3 sets of expected coordinates and the obtained coordinated.

Note

The framework must be initialized before calling sf_touch_panel_v2_api_t::calibrate API.

For ex ample:

Consider a scenario where a touch panel with resolution 480 x 272 is being used. The touch panel is touched at upper left, upper right and lower right and the obtained coordinates (x, y) when touched at these touch panel locations and their expected coordinates are as illustrated below:

Table 3 Touch Panel Coordinates

Location on Touch Panel	Expected Coordinates	Obtained Coordinates
Upper left corner	(0, 0)	(17, 20)
Upper right corner	(480, 0)	(464, 17)
Lower right corner	(480, 272)	(463, 258)

These 3 sets of values (expected and obtained coordinates) are passed into the calibrate function in the expected format. Once the sf_touch_panel_v2_api_t::calibrate API function is called, the required calibration factors are calculated and stored in the control structure. The touch data now obtained will be calibrated.

Note

Once the sf_touch_panel_v2_api_t::close API function is called the stored calibration factors are erased and the calibrate API must be called again to obtain the calibrated touch data.

Create a Custom Touch Panel Chip Driver

To create your custom Touch Chip Driver, refer to the existing touch chip driver code in SSP which is located at synergy/ ssp_supplemental / touch_drivers .

Note

*This directory will only be visible if any existing touch chip driver touch_panel_chip_ft5x06 or touch_panel_chip_sx8654 is selected in the Synergy configurator . To do this go to the Synergy configurator and select **New** > **Framework** > **Input** > **Touch Panel V2 Framework on sf_touch_panel_v2** > **Add Touch Driver**. Select any existing touch driver for reference and generate project content .*
Refer to the SSP Module Development Guide document to learn more about creating a custom driver. The document can be found using this
link: https://www.renesas.com/us/en/document/apn/ssp-module-development-guide-application-note.

The following instructions can be used to connect a custom touch chip driver to the Touch Panel V2 Framework:

- Implement Touch Chip Driver Instance (synergy/ssp_supplemental/inc/framework/instances) and source file (synergy/ssp_supplemental/touch_drivers).
- Make sure the chip specific settings are configured based on the touch screen used and the applications use case.
- Implement the payloadGet function to obtain touch events and touch coordinates from the Touch Panel Controller.
- Implement a reset function reset to reset the touch chip.
- If the touch chip is using communication protocol other than I2C or if it doesn't support external IRQ then modify the open, payloadGet, reset and close functions. Also the touch chip driver Configuration XML (located under .module_descriptions folder) must be modified.

Configure the Custom Touch Panel Chip Driver

To configure a custom touch chip driver to a project, follow the steps below:

Step 1. Create a Synergy C project.

Step 2. Update existing touch driver XML for the custom touch driver by following these steps. Modify any existing touch XML as described below:

- Go to the .module_descriptions folder under the project root folder.
- Edit touch chip driver XML:
    - Renesas##HAL Drivers##touch panel##touch_panel_chip_ft5x06####<version>.xml Or
    - Renesas##HAL Drivers##touch panel##touch_panel_chip_sx8654####<version>.xml
- Change name of the instance structure with name of custom driver instance structure. For example "g_touch_panel_chip_xxxxx"
- Add the new instance declaration and api declaration in <header> tab. For example,

```
    <header>
  extern const sf_touch_panel_chip_instance_t g_touch_panel_chip_xxxxx;
  extern const sf_touch_panel_chip_api_t g_sf_touch_panel_chip_xxxxx;
    </header>
```

- Rename all touch chip number in the XML with custom touch chip number.
- If communication protocol other than I2C is used then change the interface and id under the <requires> tag. For example,

```
    <requires>
  id="module.external.ex_touch_panel_chip_ft5x06.requires.spi " interface="
interface.framework . sf_spi_v2_on_sf_spi " display="Add framework"
    </requires >
```

- Save and close the XML file

Step 3. Open project configurator and add Touch panel V2 framework, by selecting **New** > **Framework** > **Input** > **Touch Panel V2 Framework on sf_touch_panel_v2** (If the configurator was already opened, close and open it again).

Step 4. Configure all components.

Step 5. Add the custom touch panel chip driver to the "Add Touch Driver" box. (The custom touch driver open will be visible if XML is modified correctly).

Step 6. Generate Project Content and add new directory (for example, touch_panel_chip_xxxxxx) structure under synergy/ ssp_supplemental/ touch_drivers, the directory structure will look like this: synergy/ ssp_supplemental/ touch_drivers/ touch_panel_chip_xxxxxx (replace xxxxxx with touch controller part no).

Step 7. Add the custom driver code into this directory (ssp_supplemental/ touch_drivers/ touch_panel_chip_xxxxxx).

Step 8. Exclude existing driver from build if any (**right-click the .c file** > **Exclude from build...** > **Select all** > **OK**).

Step 9. Build the code.

Touch Panel V2 Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.26.4 Including the Touch Panel V2 Framework Module in an Application

This section describes how to include the Touch Panel V2 Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Touch Panel V2 Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Touch Panel V2 Framework module is g_sf_touch_panel_v2_0. This name can be changed in the associated Properties window.)

Touch Panel V2 Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_touch_panel_v2_0 Touch Panel V2 Framework on sf_touch_panel_v2	Threads	**New Stack**> **Framework**> **Input**> **Touch Panel V2 Framework on sf_touch_panel_v2**

When the Touch Panel V2 Framework module on sf_touch_panel_v2 is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

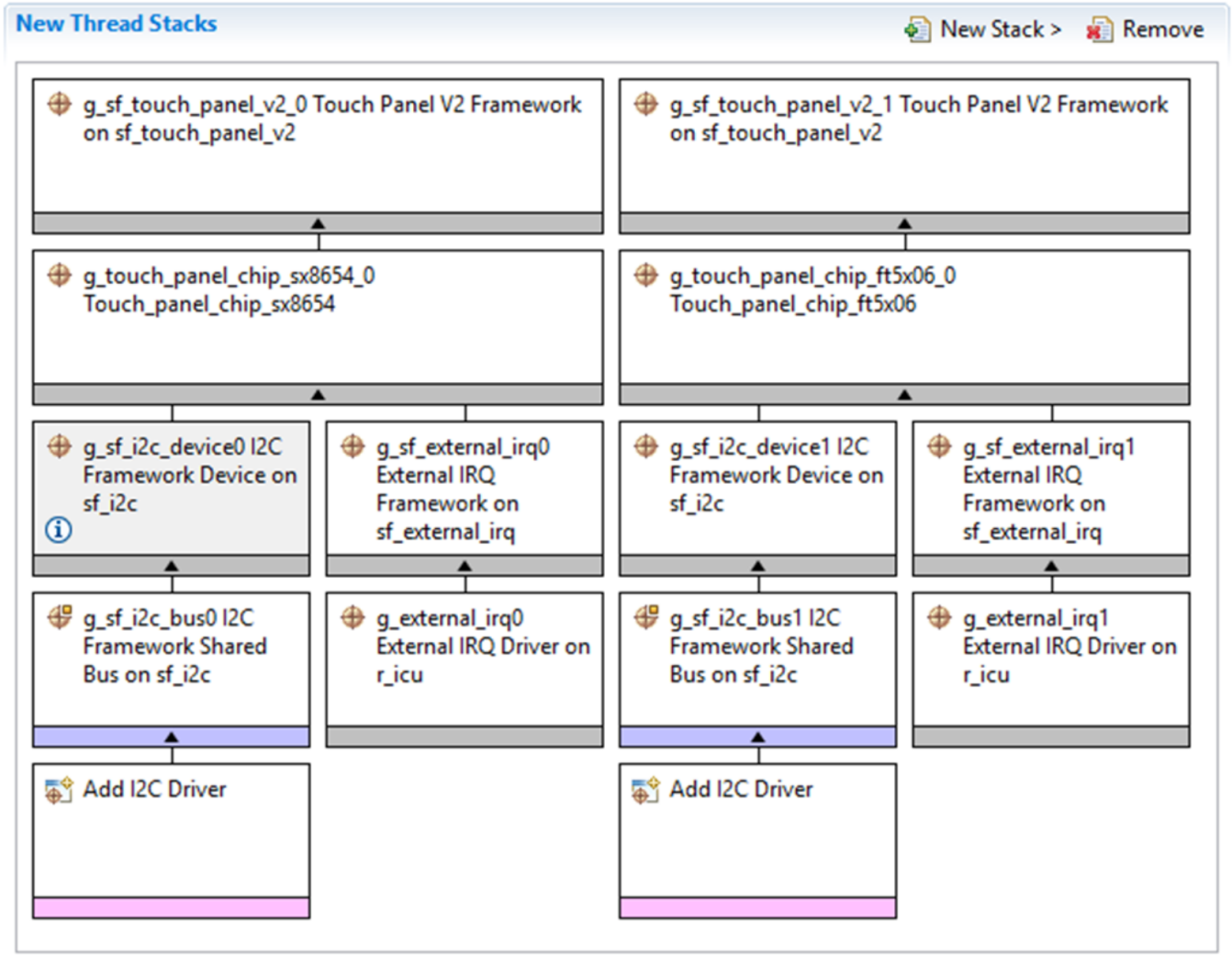


Figure 215: Touch Panel V2 Framework Module Stack

4.1.26.5 Configuring the Touch Panel V2 Framework Module

The Touch Panel V2 Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it

much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Touch Panel V2 Framework Module on sf_touch_panel_v2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Thread Stack Size	512	Specify the touch panel thread stack size.
Name	g_sf_touch_panel_v2_0	Module name.
Thread Priority	3	Specify the thread priority.
Update Hz	10	Specify the update rate in Hertz.
Touch Coordinate Rotation Angle (Clockwise)	0, 90 (Select this if 'Screen Rotation Angle' is 'CW' or '90'), 180 (Select this if 'Screen Rotation Angle' is FLIP or '180'), 270 (Select this if 'Screen Rotation Angle' is 'CCW' or '270') Default: 0	Select the touch coordinate rotation angle.
Name of generated initialization function	sf_touch_panel_v2_init0	Specify the name of the generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Select if sf_touch_panel_v2 will be initialized during startup.
Auto Start	Enable, Disable Default: Enable	Enabling this will start to get the touch data.
Name of touch panel callback function to be defined by user	NULL	Touch panel callback function name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated Properties window from the SSP configurator.

Configuring the Touch Panel V2 Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules:

Configuration Settings for the Touch Panel Chip sx8654

ISDE Property	Value	Description
Name	g_touch_panel_chip_sx8654_0	Module name.
Horizontal pixel count	480	Specify the number of horizontal pins.
Vertical pixel count	272	Specify the number of vertical pixels.
Reset Port	00:11 Default: 07	Select the chip reset port.
Reset Pin	00:15 Default: 11	Select the chip reset pin.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Touch Panel Chip ft5x06

ISDE Property	Value	Description
Name	g_touch_panel_chip_ft5x06	Module name.
Horizontal pixel count	800	Specify the number of horizontal pins.
Vertical pixel count	480	Specify the number of vertical pixels.
Reset Port	00:11 Default: 10	Select the chip reset port.
Reset Pin	00:15 Default: 02	Select the chip reset pin.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Framework Device on sf_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	Selects if code for parameter checking is to be included in the build.
Name	g_sf_i2c_device0	Give a name to identify the I2C Framework device. API, Config and Control instances will be created based on this name.
Slave Address	0x00	Specify the address of the I2C slave device.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Select the I2C address mode.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Shared Bus on sf_i2c

ISDE Property	Value	Description
Name	g_sf_i2c_bus0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master Driver on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_i2c0	Module name.
Channel	0	Specify the IIC channel to be used with this configuration.
Rate	Standard, Fast-mode, Fast-mode Plus Default: Standard	Standard, Fast, and Fast-plus. (See IIC Rate Calculation.)

Slave Address	0x00	Set the address of the slave device the I2C master will be communicating with.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
Timeout Mode	Short Mode, Long Mode Default: Short Mode	Select the timeout mode.
Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the receive interrupt priority.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the transmit interrupt priority.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the transmit end interrupt priority.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Select the error interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Fixed	Select the transfer size.
Source Address Mode	Incremented	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode)	Source	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event IIC0 TXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Incremented	Select the transfer size.
Source Address Mode	Fixed	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Destination	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event IIC0 RXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master Driver on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_i2c0	Module name.
Channel	0 to 9	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7: 0 1 2 3 4 5 6 7 8 9; Series S3 : 0 1 2 3 4 - - - - 9; Series S1 : 0 1 - - - - - - - 9.
Rate	Standard, Fast-mode, Fast-mode plus Default: Standard	Select the I2C data rate.
Slave Address	0x00	Specify the slave address.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.

Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the receive interrupt priority.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit interrupt priority.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit end interrupt priority.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the error interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.

Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer0	Module name.
Mode	Block	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Fixed	Select the transfer size.
Source Address Mode	Incremented	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Source	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Incremented	Select the transfer size.
Source Address Mode	Fixed	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Destination	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event SCI0 RXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ Framework on sf_external_irq

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_sf_external_irq0	Framework name.
Event	None, Semaphore Put Default: Semaphore Put	Event selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ Driver on r_icu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling, Rising, Both Edges, Low Level Default: Rising	Configures edge or level triggering.
Digital Filtering	Enabled, Disabled Default: Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64 Default: PCKL/64	Sets noise filter sampling period.

Interrupt enabled after initialization	True, False Default: True	Determines if the interrupt is enabled immediately after initialization.
Callback	NULL	A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Pin Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	An Interrupt priority can be registered in external_irq_cfg_t::irq_ipl.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Touch Panel V2 Framework Module Clock Configuration

An example implementation of the I2C interface uses the SCI peripheral and is described here. Other implementation choices might have different selections and can be inferred from the following example. The SCI peripheral module uses the PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator **Clock** tab prior to a build, or by using the CGC Interface at run-time. During configuration, the I2C transfer rate is calculated and set internally by the driver based on the user selected PCLB rate and the user selected transfer rate. If the PCLKB is configured in such a manner that the user selected rate cannot be achieved, an error will be returned when initializing the driver.

Touch Panel V2 Framework Module Pin Configuration

An example implementation of the I2C interface uses the SCI peripheral and is described here. Other implementation choices might have different selections, but can be inferred from the below example. Synergy Kit specific settings for pins are shown in the following section. The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins:

Pin Selection Sequence for the Touch Panel V2 Framework Module

Resource	ISDE Tab	Pin selection Sequence

SCI0	Pins	Select **Peripherals**> **Connectivity:SCI**> **SCI0**

Note

The selection sequence assumes the SCI0 is the desired hardware target of the driver.

Pin Configuration Settings for the DAC Driver on r_dac

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled Simple I2C	Select Simple I2C as the Operation Mode for SPI on SCI.
RXD1_SCL1_MISO1	None, P212, P708 Default: None	SCL pin.
TXD1_SDA1_MOSI1	None, P213, P709 Default: None	SDA pin.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.26.6 Using the Touch Panel V2 Framework Module in an Application

The typical steps in using the Touch Panel V2 Framework module in an application are:

1. Register a user callback in application code to receive touch data (if needed).
2. Initialize the touch panel V2 framework module using the sf_touch_panel_v2_api_t::open API function (done automatically within SSP if "auto initialization" property is enabled).
3. Calibration of touch data can be done by using the sf_touch_panel_v2_api_t::calibrate API function (If required).
4. Start the touch panel framework module to begin scanning the touch data from the touch chip using the sf_touch_panel_v2_api_t::start API function (done automatically within SSP if the "auto start" property is enabled).
5. The registered user callback is invoked by the framework when a touch event occurs and is processed in application code.
6. Get the data using the sf_touch_panel_v2_api_t::touchDataGet API function (if the user callback is not registered).
7. Operate on the received touch data as needed using application code.
8. Stop receiving the touch data using the sf_touch_panel_v2_api_t::stop API function.
9. Close the module using the sf_touch_panel_v2_api_t::close API function (if required).

These common steps are illustrated in a typical operational flow in the following figure:

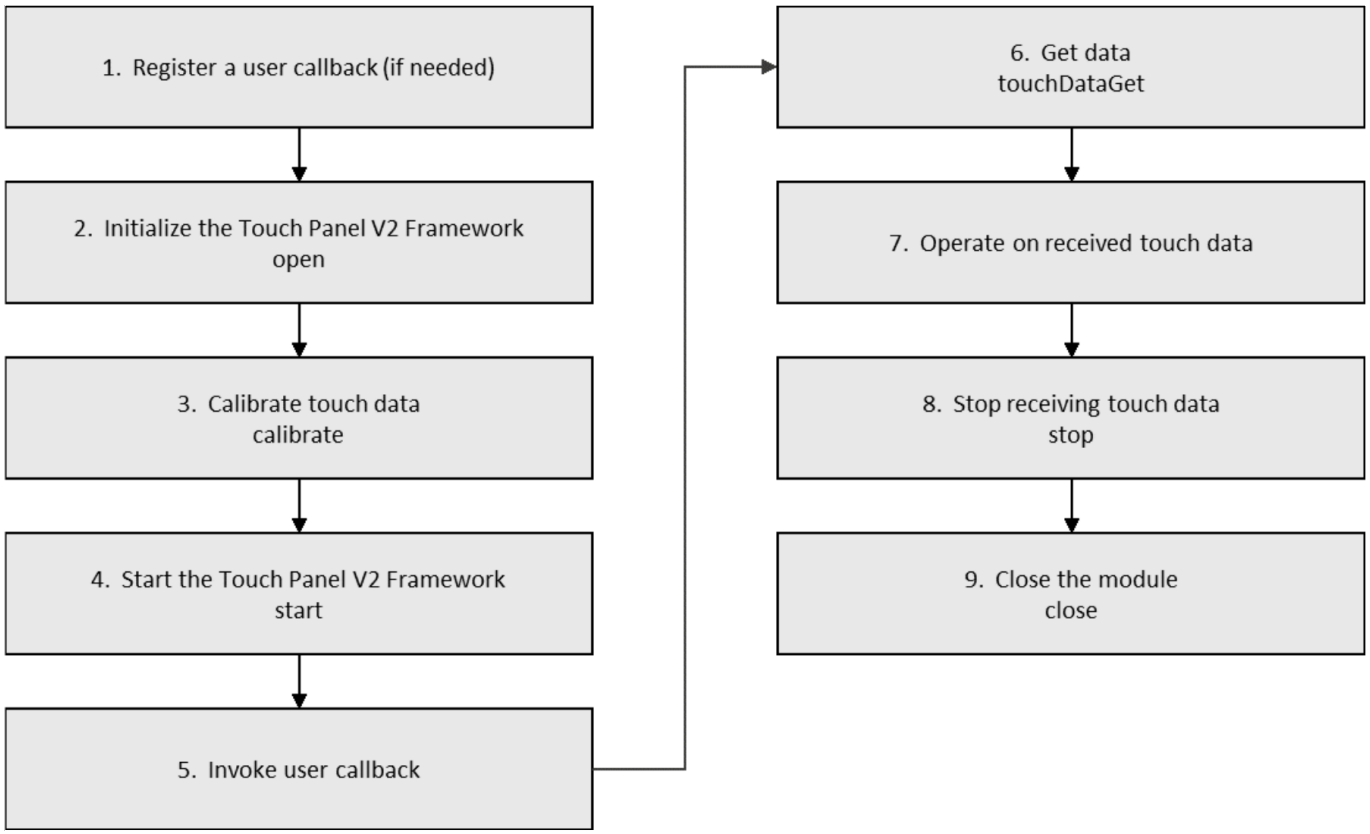


Figure 216: Flow Diagram of a Typical Touch Panel V2 Framework Module Application

4.1.27 UART Communications Framework

4.1.27.1 UART Communications Framework Module Introduction

The UART communications framework implements a high-level API for serial communications supporting the industry standard UART protocol on a UART-compliant Synergy MCU peripheral. It utilizes the r_sci_uart HAL driver to configure and operate the Synergy MCU SCI peripheral in the UART mode.

UART Communications Framework Module Features

This module is a ThreadX-aware communications framework; it uses ThreadX objects to ensure that the operations are thread safe.

Key features include:

- Support for UART Communications protocol
- Support for locking a channel to reserve exclusive access
- ThreadX-aware implementation

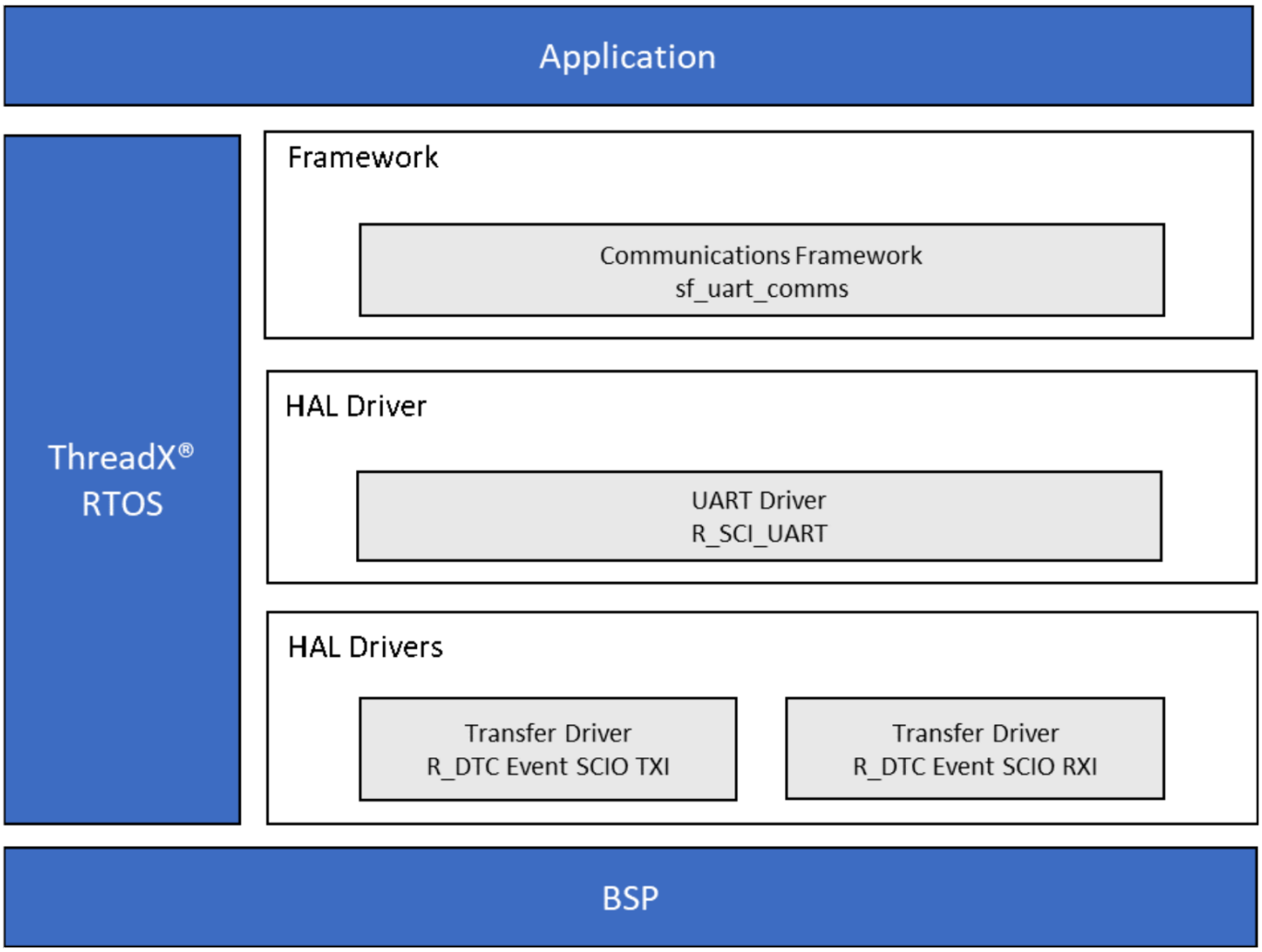


Figure 217: UART Communications Framework Module Block Diagram

4.1.27.2 UART Communications Framework Module APIs Overview

The UART Communications Framework module defines APIs for opening, closing, reading and writing to the communications channel. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

UART Communications Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_comms0.p_api->open(g_sf_comms0.p_ctrl, g_sf_comms0.p_cfg); Initialize communications driver.
close	g_sf_comms0.p_api->close(g_sf_comms0.p_ctrl); Clean up communications driver.

read	g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl, &destination, bytes, timeout); Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.
write	g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl, &source, bytes, timeout); Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.
lock	g_sf_comms0.p_api->lock(g_sf_comms0.p_ctrl, lock_type, timeout); Lock the communications driver. Reserve exclusive access to the communications driver.
unlock	g_sf_comms0.p_api->unlock(g_sf_comms0.p_ctrl, lock_type); Unlock the communications driver. Release exclusive access to the communications driver.
versionGet	g_sf_comms0.p_api->version(&version); Store the driver version in the provided version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Channel opened successfully.
SSP_ERR_IN_USE	Channel already in use.
SSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure.
SSP_ERR_QUEUE_UNAVAILABLE	Cannot open transmit or receive queue or both.
SSP_ERR_INTERNAL	Internal error occurs.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.

SSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
SSP_ERR_OVERFLOW	Hardware overflow.
SSP_ERR_FRAMING	Framing error.
SSP_ERR_PARITY	Parity error.
SSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.27.3 UART Communications Framework Module Operational Overview

The UART Framework provides an easy-to-use communication framework using the standard UART protocol. In addition to the high-level API functions to read and write data from the UART device in a thread safe manner, the framework also provides API functions for applications to lock (and unlock) the UART channel to a thread. This is particularly useful when multiple application threads try to communicate with the same UART device and a context switch could upset the high-level application protocols and/or state machines implemented on top of the UART (like Kermit, for example).

UART Communications Framework Module Important Operational Notes and Limitations

UART Communications Framework Module Operational Notes

- The UART Framework module is reentrant for any channel.
- The UART Framework uses the UART Driver on r_sci_uart module for communicating with a UART device. The UART Driver can be augmented by adding DTC drivers (using the Synergy Platform configurator in the ISDE) to perform read or write transactions with a UART device without interrupting the CPU. When the UART Framework is used to read data from a UART device, it will rely on the UART Driver's callback feature to read data and will not use the DTC (even if the DTC module support is added to the UART Driver through the configurator). This is done to avoid any potential timing and synchronization issues that could arise when the driver uses the DTC to read data from the device. When using the UART Framework to write data to a UART device, it will use the DTC to perform the transaction (if the driver is configured to use the DTC).

UART Communications Framework Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.27.4 Including the UART Communications Framework Module in an Application

This section describes how to include the UART Communications Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the UART Communications Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the UART

Communications Framework is g_sf_comms0. This name can be changed in the associated Properties window.)

UART Communications Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_comms0 Communications Framework on sf_uart_comms	Threads	**Framework** > **Connectivity** > **Communications Framework on sf_uart_comms**

When the UART Communications Framework on sf_uart_comms is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

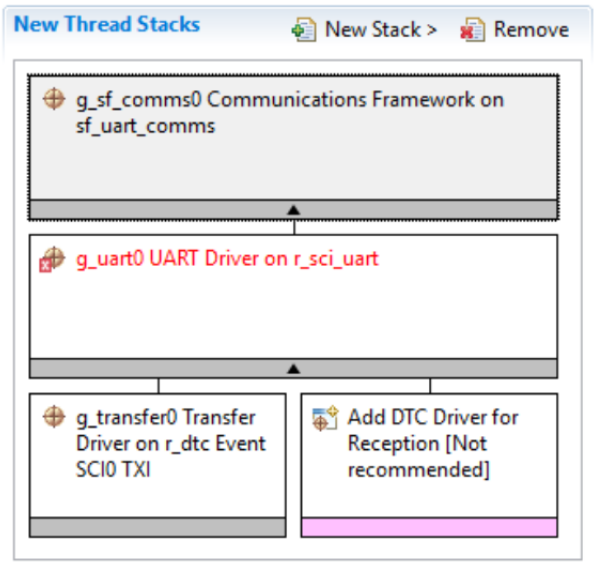


Figure 218: UART Communications Framework Module Stack

4.1.27.5 Configuring the UART Communications Framework Module

The UART Communications Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the UART Communications Framework Module on sf_uart_comms

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if parameter checking is included.
Read Input Queue Size (4-Byte Words)	15	Buffer size for data reception queue. sf_uart_comms utilizes the ThreadX Queue for the queue management.
Name	g_sf_comms0	Name of UART communications framework module.
Name of generated initialization function	sf_comms_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the UART Communications Framework Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the UART HAL Module on r_sci_uart

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to "Enable" and specify the configuration "Name of UART callback function for the RTS external pin control".

Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to "Disable" reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting "Disable" to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set "Disable" to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select "CTS" for this configuration parameter and enable the configuration "External RTS Operation" specifying the configuration "Name of UART callback function for the RTS external pin control".
Name of UART callback function to be defined by user	NULL	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Defaualt: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Fallling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set "Falling Edge" to this configuration.
Noise Cancel	Enable, Disable Defaualt: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Defaualt: Enable	Bit rate modulation enable selection.
Receive FIFO Trigger Level	One, Max Default: Max	Receive FIFO trigger level selection.

Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selecition.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Error interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Defaualt: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.

Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.

Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

UART Communications Framework Module Clock Configuration

The UART Communications Framework has no specific clock configuration requirements.

UART Communications Framework Module Pin Configuration

The UART Communications Framework uses pins on the MCU to communicate to external devices based on the lower level implementation selected. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the lower level implementation pins.

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the UART Communications Framework Module on sf_uart_comms

Resource	ISDE Tab	Pin selection Sequence
UART	Pins	Select **Peripherals**> **Connectivity: SCI**> **SCI8**

Note

The selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for the UART Communications Framework Module on sf_uart_comms

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only	Pin group selection.

Operation Mode	Disabled, Custom, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Asynchronous UART as the Operation Mode for a UART Receiver implementation.
TXD_MOSI	None, PB05, P105 Default: None	TXD pin P105.
RXD_MISO	None, PB05, P104 Default: None	RXD pin P104.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

Pin Selection Sequence for Communications Framework on UART, USB or Telnet (Transmitter)

Resource	ISDE Tab	Pin selection Sequence
UART	Pins	Select **Peripherals** > **Connectivity: SCI**> **SCI3**

Note

These selection sequences are examples for selected implementations. Others are also possible depending on the target hardware.

Pin Configuration Settings for UART

Pin Configuration Property	Settings	Description
Pin Group Selection	Mixed, _A Only, _B Only	Pin group selection.
Operation Mode	Disabled, Custom, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Asynchronous UART as the Operation Mode for a UART Transmitter implementation.
TXD_MOSI	None, P707, P409 Default: None	TXD pin P707.
RXD_MISO	None, P706, P408 Default: None	RXD pin P706.

Note

These selection sequences are examples for selected implementations. Others are also possible depending on the target hardware.

4.1.27.6 Using the UART Communications Framework Module in an Application

The steps in using the UART Communications Framework module on sf_audio_record_adc in a typical application are:

1. Initialize the UART Communications Framework using the sf_comms_api_t::open API.
2. Lock the channel for continuous communications using the sf_comms_api_t::lock API if needed.
3. Receive data using the sf_comms_api_t::read API.
4. Send data using the sf_comms_api_t::write API.
5. Unlock the channel from continuous communication using the sf_comms_api_t::unlock API if needed.
6. Close the channel using the sf_comms_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

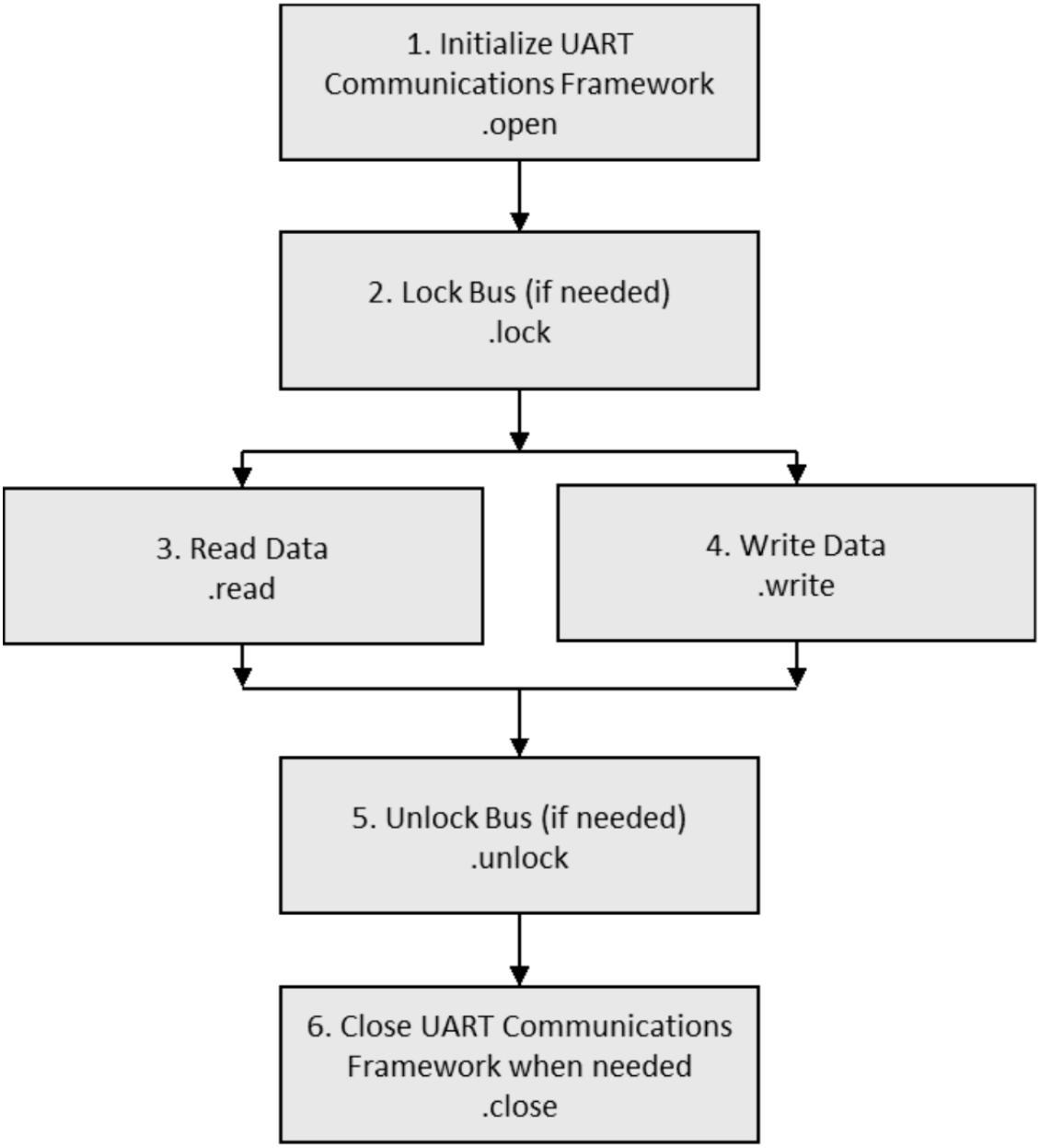


Figure 219: Flow Diagram of a Typical UART Communications Framework Module Application

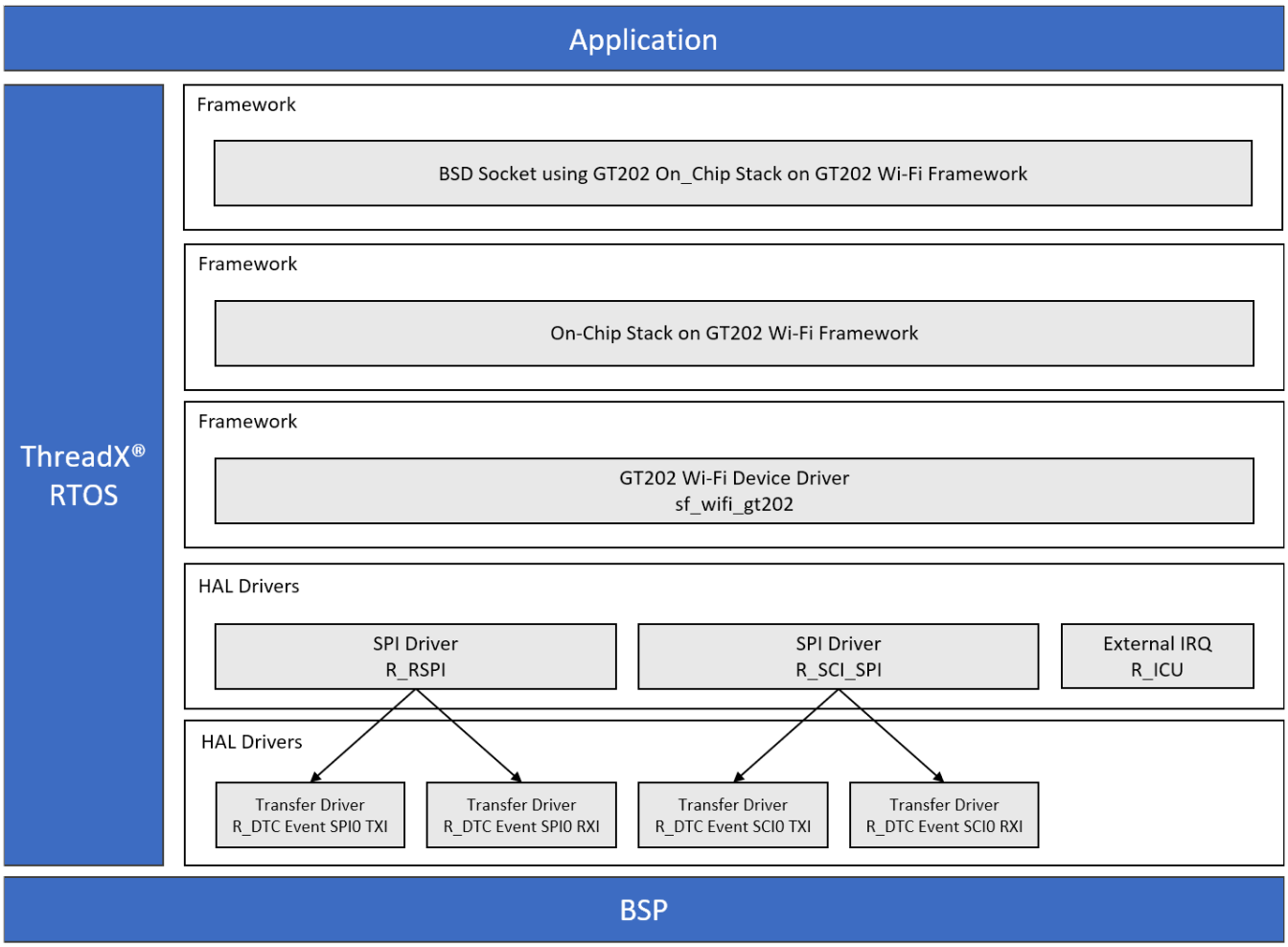
4.1.28 Wi-Fi Framework

4.1.28.1 Wi-Fi Framework Introduction

The Wi-Fi framework provides a high-level API for configuring and provisioning Wi-Fi modules as well as performing data transfers with or without on-chip networking capability. Currently, only the Qualcomm GT202 module is supported. The Wi-Fi framework communicates through the SPI with the underlying GT202 module.

Wi-Fi Framework Module Features

- Provides high-level APIs to configure and provision a Wi-Fi module
- Provides four different implementations for:
    - A Wi-Fi device driver stack using the sf_wifi_gt202 framework.
    - An on-chip stack using the sf_wifi_onchip_stack framework.
    - A BSD socket stack using the sf_wifi_onchip_stack framework.
    - A NetX and NetX Duo port using the sf_wifi_nsal_nx framework.
- Using NetX and NASL:
    - Allows the same application code to be used across different Wi-Fi modules.
    - Allows for easy migration of the Ethernet-based application to a Wi-Fi based application.
    - Allows for debugging and fine-tuning the application and TCP/IP stack as required by the application.
    - The current NSAL implementation only provides NetX NSAL. Adding support for a new network stack requires implementing the appropriate NSAL.
- Using the On-chip networking stack:
    - Is beneficial when using MCUs with a small memory footprint.
    - Provides a BSD sockets interface to create socket-based applications with the On-chip TCP/UDP.
- Provides an option to integrate 3rd-party application protocols on top of TCP/IP such as MQTT and COAP without using the NetX stack.
- Provides support for Wi-Fi Protected Setup (**WPS**) router configuration using Push-Button and PIN methods.


Application
ThreadX®
RTOS
Framework
BSD Socket using GT202 On_Chip Stack on GT202 Wi-Fi Framework
Framework
On-Chip Stack on GT202 Wi-Fi Framework
Framework
GT202 Wi-Fi Device Driver
sf_wifi_gt202
HAL Drivers
SPI Driver
R_RSPI
SPI Driver
R_SCI_SPI
External IRQ
R_ICU
HAL Drivers
Transfer Driver
R_DTC Event SPI0 TXI
Transfer Driver
R_DTC Event SPI0 RXI
Transfer Driver
R_DTC Event SCI0 TXI
Transfer Driver
R_DTC Event SCI0 RXI
BSP

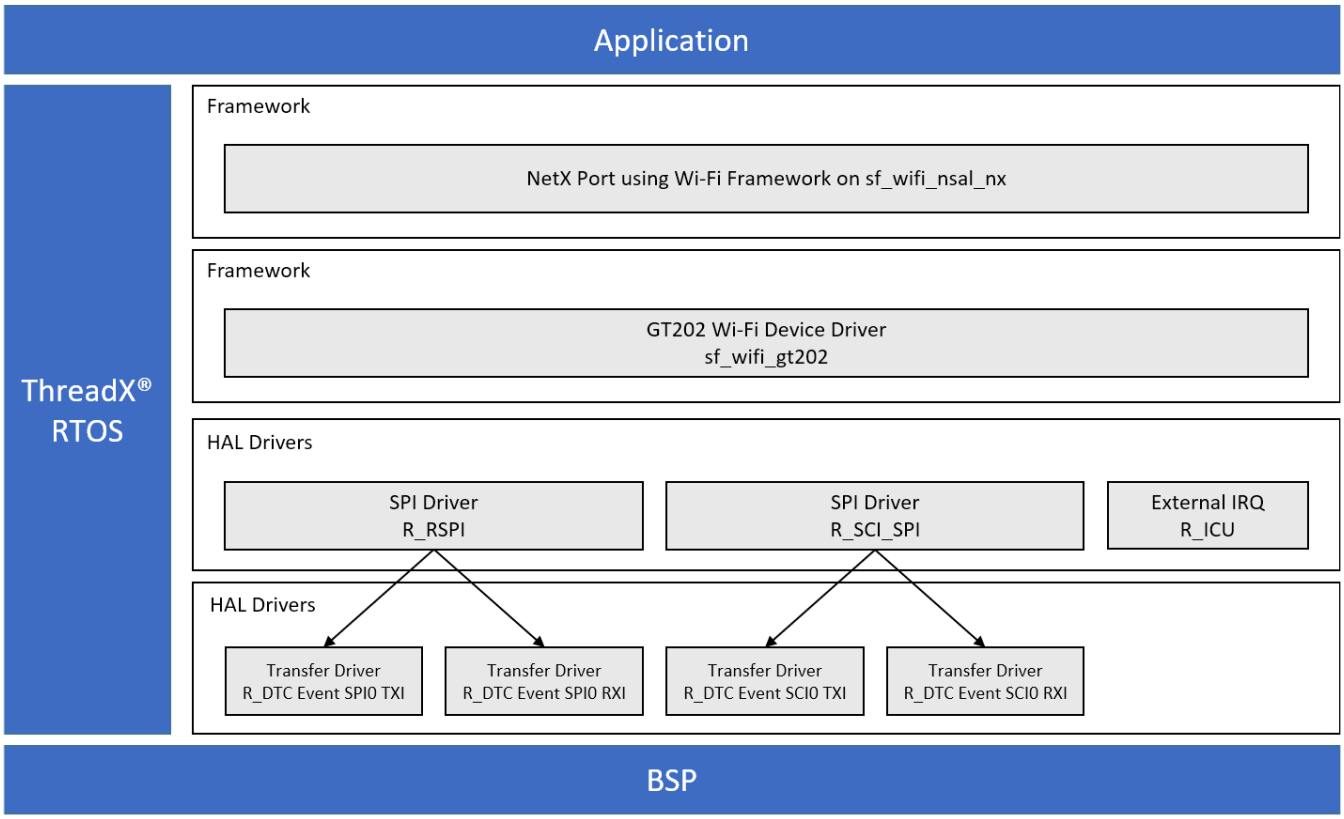

Application
ThreadX®
RTOS
Framework
NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx
Framework
GT202 Wi-Fi Device Driver
sf_wifi_gt202
HAL Drivers
SPI Driver
R_RSPI
SPI Driver
R_SCI_SPI
External IRQ
R_ICU
HAL Drivers
Transfer Driver
R_DTC Event SPI0 TXI
Transfer Driver
R_DTC Event SPI0 RXI
Transfer Driver
R_DTC Event SCI0 TXI
Transfer Driver
R_DTC Event SCI0 RXI
BSP

Figure 220: WiFi Framework Module Block Diagram

Note

The On-Chip Stack on GT202 Wi-Fi Framework can be used as a lower-level implementation of the BSD Socket Framework or on its own. The GT202 Wi-Fi Device Driver and its lower-level modules is included below all other Wi-Fi Framework implementations.

4.1.28.2 Wi-Fi Framework Module APIs Overview

The Wi-Fi Framework defines API functions for each of the related modules. The following descriptions explain the operation of each API. A table of common return codes follows at the end of the section. Refer to the API reference section for the associated module for additional details.

Additionally, a more detailed API description, along with a working example application (which is too lengthy to include in this document), is available on the Renesas web site. Just search for the associated application note document number, r11an0226eu, in the top page search bar on www.Renesas.com. It is highly recommended that you use the application note to augment the summary descriptions found in this document.

Wi-Fi Framework Module APIs

WiFi Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_wifi0.p_api->open (g_sf_wifi0.p_ctrl, g_sf_wifi0.p_cfg); This API initializes and enables the Wi-Fi module. The open function returns the Wi-Fi control structure, uniquely identifying the instance of the Wi-Fi framework.
close	g_sf_wifi0.p_api->close(g_sf_wifi0.p_ctrl); This API un-initializes the Wi-Fi module and powers it off.
infoGet	g_sf_wifi0.p_api->infoGet (g_sf_wifi0.p_ctrl, wifi_info); This API takes the WiFi control structure as an argument and returns the following information obtained from the WiFi module: - Chipset/driver information string - RSSI value (unsigned integer 16 bits) - Noise level (unsigned integer 16 bits) - Link Quality (unsigned integer 16 bits)

statisticsGet	g_sf_wifi0.p_api->statisticsGet (g_sf_wifi0.p_ctrl, p_stats); This API gets the data statistics from the Wi-Fi module. It takes the Wi-Fi control structure as an argument and returns the following statistics: - Received packets (unsigned integer 32 bits) - Transmitted packets (unsigned integer 32 bits) - Transmit packet errors (unsigned integer 32 bits)
transmit	g_sf_wifi0.p_api->transmit (g_sf_wifi0.p_ctrl, p_buffer, length); This API sends the data/packet out. This function takes the Wi-Fi control structure as an argument. It takes the network packet buffer, and the network packet buffer length as arguments. The Wi-Fi framework transmit function passes the packet buffer to the Wi-Fi driver for transmission.
provisioningGet	g_sf_wifi0.p_api->provisioningGet (g_sf_wifi0.p_ctrl, &sf_wifi_provisioninfo); This API takes the Wi-Fi control structure as an argument and returns the following parameters: - Mode (enumeration, that is, AP or client) - Channel (unsigned integer 8 bits) - SSID (string) - Security type (enumeration) - Encryption type (enumeration) - Security key (string)
provisioningSet	g_sf_wifi0.p_api->provisioningSet (g_sf_wifi0.p_ctrl, &g_sf_wifi_provisioninfo); This API sets the Wi-Fi module in the given mode AP/Station. The provisioningSet function uses the following parameters to provision the WiFi module: - Mode (enumeration, that is, AP or client) - Channel (unsigned integer 8 bits), only used in AP mode - SSID (string) - Security type (enumeration) - Encryption type (enumeration) - Security key (sting). - Connection Status Notification Callback function: Used to get connection status change notification *See note at the end of this table.

scan	g_sf_wifi0.p_api->scan (g_sf_wifi0.p_ctrl, p_scan, p_count); This API scans the available SSIDs (that is, access points) in range. This function takes the Wi-Fi control structure as an argument and returns a list of SSIDs scanned by the WiFi module with the following parameters: - HW mode (enumeration a/b/g/n) - RSSI (unsigned integer 16 bits) - SSID (string) - BSSID (byte array of size 6 bytes) - Channel (unsigned integer 8 bits) - Security type (enumeration) - Encryption type(enumeration) - BSS type (enumeration) The Wi-Fi framework scan function takes the SSID count as an argument, which acts as an in/out parameter. It specifies the size of the scan result array and the Wi-Fi framework sets it to count the indicating number of scan results stored in the array.
ACLAdd	g_sf_wifi0.p_api->ACLAdd (g_sf_wifi0.p_ctrl, peer_device_mac); This API adds the given MAC address to the access control list. This function takes the Wi-Fi control structure and the MAC address as arguments.
ACLDelete	g_sf_wifi0.p_api->ACLDelete (g_sf_wifi0.p_ctrl, peer_device_mac); This API deletes the given MAC address from the access control list. This function takes the Wi-Fi control structure and MAC address as arguments.
multicastListAdd	g_sf_wifi0.p_api->multicastListAdd (g_sf_wifi0.p_ctrl, p_mac_addr); This API adds the given Multicast IP address to the multicast filer list. This function takes the Wi-Fi control structure and MAC address as arguments.
multicastListDelete	g_sf_wifi0.p_api->multicastListDelete (g_sf_wifi0.p_ctrl, p_mac_addr); This API deletes the given Multicast IP address from the multicast filer list. This function takes the Wi-Fi control structure and MAC address as arguments.
macAddressGet	g_sf_wifi0.p_api->macAddressGet (g_sf_wifi0.p_ctrl, p_mac); This API reads MAC address from WiFi module. This function takes the Wi-Fi control structure as argument and returns MAC address read from Wi-Fi module.

macAddressSet	g_sf_wifi0.p_api->macAddressSet (g_sf_wifi0.p_ctrl, p_mac); This API sets Wi-Fi module's MAC address. This function takes the Wi-Fi control structure and MAC address as arguments.
wpsStart	g_sf_wifi0.p_api->wpsStart(g_sf_wifi0.p_ctrl, &wps_data); This API starts WPS on device. The wpsStart function uses the following parameters passed to a structure to start WPS on device: - WPS method as Push-button or Pin - WPS pin. Used only with WPS pin method - WPS timeout value in seconds - Pointer to callback function to be called on change in client's connection status with AP or client connected/disconnected

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.
Provisioning: When the device is provisioned in client mode, the callback will be called when a connection with the AP is lost and then reestablished. When the device is provisioned in AP mode, this callback is called when any client connects or disconnects with the AP. When the device is provisioned in client mode, arguments passed to callback will have only the below valid field,

- event = SF_WIFI_EVENT_AP_CONNECT or SF_WIFI_AP_DISCONNECT

When device is provisioned in AP mode then arguments passed to callback will have only below valid fields,

- event = SF_WIFI_EVENT_CLIENT_CONNECT or SF_WIFI_CLIENT_DISCONNECT
- mac_addr = MAC address of client.

While calling the sf_wifi_api_t::provisioningSet API function to provision the device in client mode, the framework will not call the callback function on the successful association with the AP or on a failure. When the device is provisioned in AP mode, if the client tries to connect with the AP using the wrong password, the callback will be called twice; first with the connected event and then immediately after with the disconnected event.

On-Chip Networking Stack Support APIs

These APIs can be used to configure the Wi-Fi module when using an on-chip networking stack, which helps to configure the IP address for the interface and start/stop the DHCP server (when configured in the AP mode).

On-Chip Networking Stack Support Wi-Fi Framework Module API Summary

Function Name	Example API Call and Description

open	g_sf_wifi_onchip_stack0.p_api->open (g_sf_wifionchip_stack0.p_ctrl, g_sf_wifi_onchip_stack0.p_cfg); This API calls the WiFi framework open API which initializes the Wi-Fi module.
close	g_sf_wifi_onchip_stack0.p_api->close(g_sf_wifi_onchip_stack0.p_ctrl); This API calls the Wi-Fi framework close API which un-initializes the Wi-Fi module.
ipAddressCfg	g_sf_wifi_onchip_stack0.p_api->ipAddressCfg (g_sf_wifi_onchip_stack0.p_ctrl, p_cfg); This API configures the IP address of the interface using an on-chip networking stack. It provides facility to configure static IP address or using DHCP.
dhcpServerStart	g_sf_wifi_onchip_stack0.p_api->dhcpServerStart (g_sf_wifi_onchip_stack0.p_ctrl, p_start_ip, p_end_ip); This API starts the DHCP server on the interface (when configured in AP mode) using on-chip networking stack. It takes the range of IP addresses to be used by DHCP server.
dhcpServerStop	g_sf_wifi_onchip_stack0.p_api->dhcpServerStop (g_sf_wifi_onchip_stack0.p_ctrl); This API stops the DHCP server.
versionGet	g_sf_wifi_onchip_stack0.p_api->versionGet (&version); Retrieves the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

BSD Socket APIs

These APIs can be used for BSD socket support using the GT202 on-chip stack implementation.

BSD Socket using GT202 On-Chip Stack Wi-Fi Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_socket0.p_api->open (g_sf_socket0.p_ctrl, g_sf_socket0.p_cfg); This API initializes the networking interface.
close	g_sf_socket0.p_api->close(g_sf_socket0.p_ctrl); This API closes the network interface.

versionGet	g_sf_socket0.p_api->versionGet (&version); Retrieves the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Additionally, this implementation includes socket APIs which are compliant with BSD APIs. These APIs can be used by the application to perform data transfer using sockets. The following APIs are available:

- socket
- close
- bind
- listen
- accept
- connect
- send
- recv
- recvfrom
- sendto
- setsockopt
- getsockopt
- select

Note

While using on chip networking stack, application can use all BSD Socket APIs, all on chip Networking Stack support APIs and few Synergy Wi-Fi framework APIs. The Synergy Wi-Fi framework APIs which application can use are provisioningSet(), provisioningGet(), scan(), macAddressGet(), macAddressSet() and infoGet()

More information is available for these APIs as described in the NetX BSD 4.3 Sockets API Compliancy Wrapper for NetX User Guide which can be found on the Synergy Gallery on the SSP page under the documentation tab in the Azure RTOS Component Documents for Renesas Synergy zip file.

Wi-Fi NSAL

The Synergy Wi-Fi framework supports the NetX/NetX-Duo Network Services Abstraction Layer. This includes the NetX/NetX-Duo driver, packet transmit and receive callback functions implementation.

NetX/NetX-Duo Driver Function

The NetX/NetX-Duo driver function takes the NetX IP instance, Wi-Fi framework instance and NSAL configuration as arguments. The NSAL configuration controls the behavior of transmit and receive callback functions. The NSAL configuration includes flags which indicates zero-copy support is enabled or disabled in transmit and receive path. The NetX/NetX-Duo driver functions implement various IP driver commands used by NetX/NetX-Duo. The interface attach command calls the Wi-Fi framework open API to initialize the Wi-Fi module. The initialize command calls the Wi-Fi framework macAddressGet API to read MAC address from the Wi-Fi module. The multicast-join command calls the Wi-Fi framework multicastListAdd API to add the given MAC address to multicast list. The multicast-leave command calls the Wi-Fi framework multicastListDelete API to delete the given MAC address from the multicast list. The Send/Broadcast command calls the Wi-Fi framework transmit API to transmit a packet.

NSAL Transmit Function

The NSAL transmit function takes the NetX IP instance, the NetX packet, the Wi-Fi framework instance and the NSAL configuration as arguments. If zero-copy support is enabled, then the same NetX packet is transferred from NetX to the Wi-Fi driver. If zero-copy is not supported, then it copies data from the NetX packet to the driver buffer. It calls the Wi-Fi framework transmit API, which passes the buffer/packet to the Wi-Fi driver for further transmission.

NSAL Receive Callback

The NSAL receive callback function takes the NetX IP instance, the packet buffer, the packet buffer length and the NSAL configuration as arguments. This callback is called from the Wi-Fi device driver. If zero-copy support is enabled, then the same NetX packet is transferred from the Wi-Fi driver to NetX. If zero-copy is not supported, then it copies data from the driver buffer to the NetX packet and then passes the NetX stack for further processing. It calls the Wi-Fi framework transmit API, which passes the buffer to the Wi-Fi driver for further transmission.

More information is available for these APIs as described in the NetX User Guide which can be found on the Synergy Gallery on the SSP page under the documentation tab in the Azure RTOS Component Documents for Renesas Synergy zip file.

Wi-Fi Framework Error Codes

The following table lists the Wi-Fi Framework specific error codes. These error codes are part of ssp_err_t.

Wi-Fi Framework Error Codes

Error Codes	Description
SSP_ERR_WIFI_CONFIG_FAILED	Configuration failed.
SSP_ERR_WIFI_INIT_FAILED	Initialization failed.
SSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed.
SSP_ERR_WIFI_INVALID_MODE	Invalid mode specified.
SSP_ERR_WIFI_FAILED	WiFi failed.
SSP_ERR_WIFI_WPS_INVALID_START_INFO	Invalid input parameters.
SSP_ERR_WIFI_WPS_MULTIPLE_PB_SESSIONS	Another Push button session is already in progress.
SSP_ERR_WIFI_WPS_WALK_TIMER_TIMEOUT	WPS Timer expired.
SSP_ERR_WIFI_WPS_M2D_RECEIVED	M2D Error code received which means Registrar is unable to authenticate with the Enrollee.
SSP_ERR_WIFI_WPS_AUTHENTICATION_FAILED	WPS authentication failed.
SSP_ERR_WIFI_WPS_CANCELLED	WPS Request was not accepted by underlying driver.
SSP_ERR_WIFI_WPS_INVALID_PIN	Invalid WPS pin.

4.1.28.3 Wi-Fi Framework Module Operational Overview

The Wi-Fi framework provides a high-level interface for the application to configure the Wi-Fi module, provision the Wi-Fi module and perform data transfers. This simplifies application development and allows the same application code to be used across different Wi-Fi modules.

The following figure provides an overview of the Synergy Wi-Fi framework layered architecture:

- The Wi-Fi framework includes the enclosed 5 blocks in the middle of the architecture graph: NSAL, Wi-Fi Framework API, Wi-Fi on-chip Stack API, BSD Socket API, and the Wi-Fi Device Driver Interface.
- The vendor-provided Wi-Fi device drivers are included in the SSP package under SSP_Supplemental.

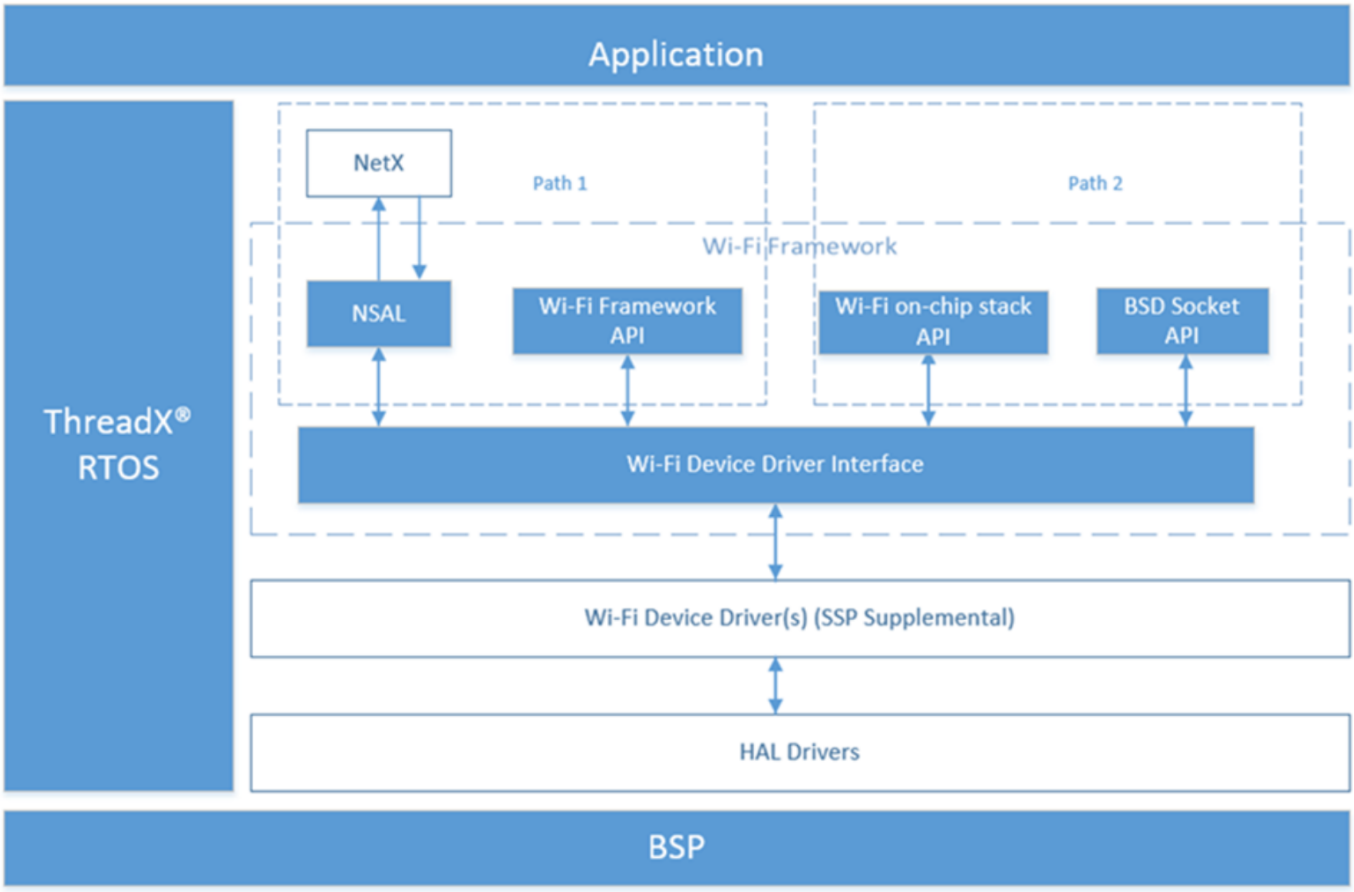


Figure 221: WiFi Framework Organization, Options and Stack Implementation

The Wi-Fi framework implementation allows Wi-Fi modules with or without On-chip networking stack support to be integrated with the SSP low-level support blocks.

- Path 1: Using NetX™/NetX Duo™, NSAL in addition to the Wi-Fi framework API blocks as shown in in the previous figure. To simplify the description, throughout this document, NetX refers to both NetX and NetX-Duo when the comment applies to both.
- Path 2: Using On-chip networking stack support API and the BSD Socket APIs as shown in the previous figure.

Wi-Fi Framework Module Important Operational Notes and Limitations

Wi-Fi Framework Module Operational Notes

- The Wi-Fi module has various parameters as specified by 802.11 standards. It is possible that individual device drivers and Wi-Fi chipsets might not support the configuration of all the functions.
- For the Wi-Fi interface to become active, at least the channel, Service Set Identifier (SSID), security scheme, and security credentials must be configured.
- The current NSAL implementation includes support for NetX (IPv4) and NetX-Duo (IPv6). NetX and NetX Duo support IPv4, however NetX Duo also supports IPv6. Adding support for a new network stack requires implementing the appropriate NSAL.
- For the security setting, WEP keys can be entered in either ASCII or Hex format and can be configured to use either 40 or 104-bit keys. A WEP key has a 24-bit initialization vector in addition to the secret key. Because of this and depending on the vendor, 64-bit WEP keys can be referred to as 40-bit keys and 128-bit WEP keys can be referred to as 104-bit keys. The Wi-Fi framework accepts 1 to 4 WEP keys of a specific format and type. In the provisioning structure, you must fill in the security type as SF_WIFI_SECURITY_TYPE_WEP and at least one (maximum is four) WEP key in the key buffer.

Wi-Fi Framework Module API Use Notes

Open:

When using the Wi-Fi framework module with NSAL, that is, with NetX/NeXDuo, the application should not call the Wi-Fi framework module sf_wifi_api_t::open API directly; instead, it should call the NetX nx_ip_create() API, which internally calls sf_wifi_api_t::open API from the NetX driver.

When using the On chip networking stack, the application should call the sf_wifi_api_t::open API from the BSD socket interface, which internally calls the Wi-Fi framework sf_wifi_api_t::open API.

Close:

When using the Wi-Fi framework module with NSAL, that is, with NetX/NetX Duo, the application should call the NetX nx_ip_delete() API directly.

When using the On chip networking stack, the application should call the sf_wifi_api_t::close API from the BSD socket interface, which internally calls the Wi-Fi framework module sf_wifi_api_t::close API.

ProvisioningSet:

When the device is provisioned in client mode, the callback will be called when the connection with the AP is lost and re-established. When the device is provisioned in AP mode, this callback is called when any client connects/disconnects with the AP. When the device is provisioned in client mode, the arguments passed to the callback will only have the following valid field:

event = SF_WIFI_EVENT_AP_CONNECT or SF_WIFI_AP_DISCONNECT

When the device is provisioned in AP mode, the arguments passed to the callback will only have the following valid fields:

event = SF_WIFI_EVENT_CLIENT_CONNECT or SF_WIFI_CLIENT_DISCONNECT

mac_addr = MAC address of client.

While calling the sf_wifi_api_t::provisioningSet API to provision the device in client mode, the framework will not call the callback function on successful association with the AP or on a failure.

When the device is provisioned in AP mode, if the client tries to connect with the AP using the wrong

password, the callback will be called twice, first with the connected event and then immediately after this, with the disconnected event.

InfoGet:

When the device is provisioned in the client mode, the RSSI value obtained using the sf_wifi_api_t::infoGet API call represents the SNR in dB. The noise_level and link_quality fields returned by the sf_wifi_api_t::infoGet API call do not contain any information - they are set to zero.

wpsStart:

If user wants to connect with Wi-Fi access-point, then user must know the SSID name, password and security type set on access-point. With this information, user can call Wi-Fi provisioning API to connect the device with given access-point. Wi-Fi Protected Setup (WPS) is a wireless network security standard that tries to make connections between a router and wireless devices faster and easier. WPS automatically configures the network name (SSID) and security key for the access point. User does not need to know the SSID and security key or passphrase when connecting WPS enabled devices.

There are 2 primary methods used in the WiFi Protected Setup:

- PIN entry – a mandatory method of setup for all WPS certified devices.
- Push button configuration – an actual push button on the hardware or through a simulated push button in the software.

Using the sf_wifi_api_t::wpsStart API, user can start WPS on device.

BSD Socket APIs:

While using the on-chip networking stack, applications can use all the BSD Socket APIs, all the On-Chip Networking Stack support APIs, and few Synergy Wi-Fi Framework APIs. The available Wi-Fi framework APIs are the sf_wifi_api_t::provisioningSet API, sf_wifi_api_t::provisioningGet API, sf_wifi_api_t::scan API, sf_wifi_api_t::macAddressGet, sf_wifi_api_t::macAddressSet API, and the sf_wifi_api_t::infoGet API.

Wi-Fi Framework Module Limitations

- The Wi-Fi framework does not support the Synergy S1 MCU Series due to memory constraints.
- Due to memory constraints, S3A6 and S128 MCUs will support only on-chip networking stack (that is, Path 2 for Wi-Fi use, where networking stack runs on Wi-Fi chipset).
- The Synergy Wi-Fi Framework APIs implemented for GT-202 are not re-entrant. All these APIs make calls to driver APIs to do the requested operation. If the GT-202 driver is working on behalf of any Wi-Fi Framework APIs and any other Wi-Fi Framework APIs are called, it will return SPP_ERR_IN_USE error until the ongoing operation is finished.
- The Synergy Wi-Fi framework sf_wifi_api_t::provisioningSet API for GT-202 may fail if the peripheral and IO pin drive capacity is not set to medium.
- While configuring the GT-202 with the SPI driver on r_rspi, set the drive capacity of the slave select pin, reset pin and SPI pins (that is, MISO, MOSI and RSPCK) to medium. While the configuring GT-202 with the SPI driver on r_sci_spi, set the drive capacity of the slave select pin, reset pin and SCI pins (that is, TXD_MOSI and RXD_MISO) to medium. Do not change the drive capacity of the SCK pin when the r_sci_spi driver is used.
- WiFi WPS functionality using GT-202 works only with WPA2 security in AP and Station mode.
- GT202 driver hangs if WPS session is terminated on peer device before session completion. However if user restarts WPS session on peer device and once WPS session is completed,

GT202 driver will exit from the loop and hanging issue will not be observed.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.28.4 Including the Wi-Fi Framework Module in an Application

This section describes how to include the Wi-Fi Framework module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Wi-Fi Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Wi-Fi Framework module is g_sf_wifi0. This name can be changed in the associated Properties window.)

Wi-Fi Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_wifi0 GT202 Wi-Fi Device Driver on sf_wifi_gt202	Threads	**New Stack> Framework> Networking> WiFi> GT202 Wi-Fi Device Driver on sf_wifi_gt202**
g_sf_wifi_onchip_stack0 On-Chip Stack on Gt202 Wi-Fi Framework	Threads	**New Stack> Framework> Networking> WiFi> On-Chip Stack on Gt202 Wi-Fi Framework**
g_sf_socket0 BSD Socket using On-Chip Stack on Gt202 Wi-Fi Framework	Threads	**New Stack> Framework> Networking> WiFi> BSD Socket using On-Chip Stack on Gt202 Wi-Fi Framework**
g_sf_el_nx0 NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx	Threads	**New Stack> Framework> Networking> WiFi> NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx**

When the Wi-Fi Framework module on sf_wifi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

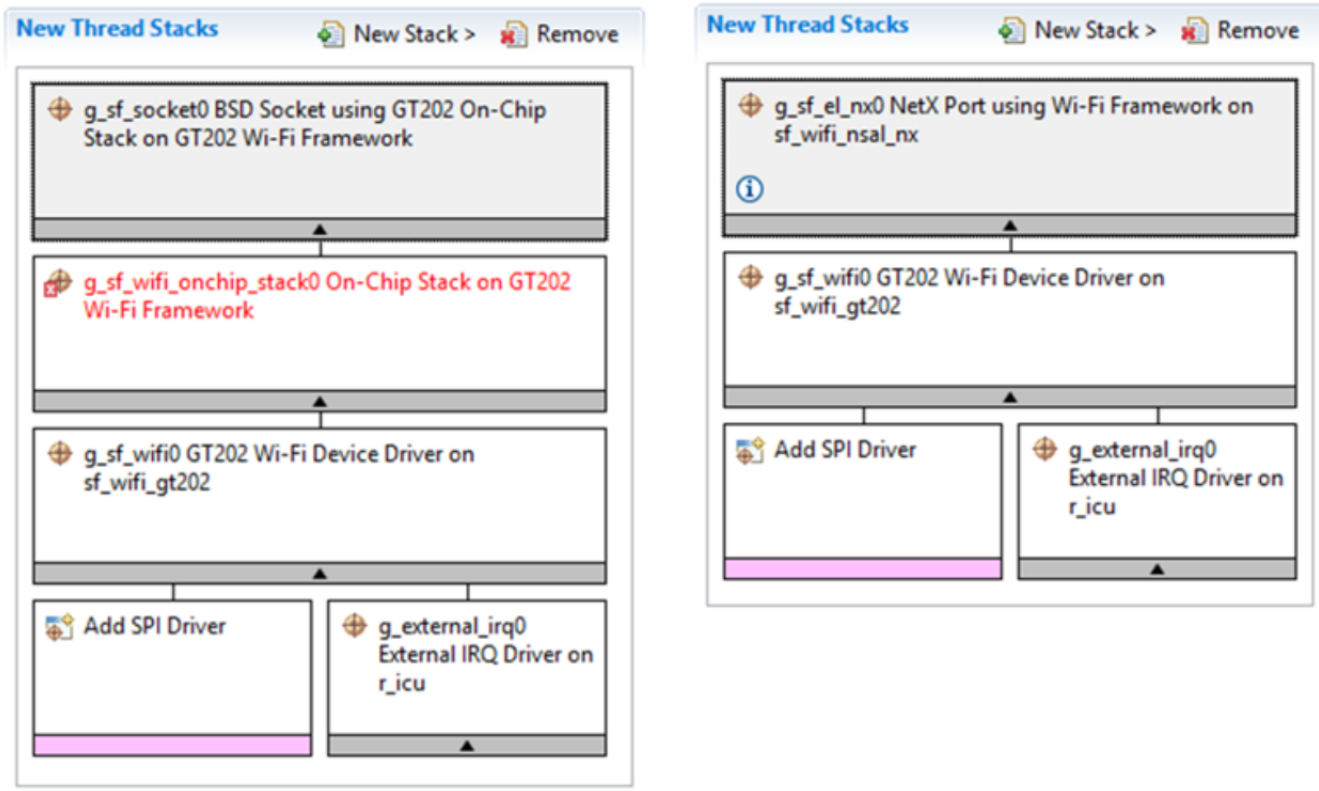


Figure 222: WiFi Framework Module Stack

4.1.28.5 Configuring the Wi-Fi Framework Module

The Wi-Fi Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the BSD Socket Using GT202 On-Chip Stack on GT202 Wi-Fi Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.

Name (Must be a valid C Symbol)	g_sf_socket0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the On-Chip Stack on GT202 Wi-Fi Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_wifi_onchip_stack0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_el_nx0	Module name.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Wi-Fi Device Driver on sf_wifi_gt202

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection.
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection.
Name (Must be a valid C symbol)	g_sf_wifi0	Module name.

Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode selection.
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection.
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled Default: Enabled	Ready/Clear to send selection.
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection.
Broadcast SSID (AP mode only)	Enabled, Disabled Default: Enabled	Broadcast SSID selection.
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection.
Station inactivity timeout in seconds (AP mode only and must be greater than 0)	100	Station inactivity timeout selection.
Requested High Throughput	Enabled, Disabled Default: Disabled	Requested high throughput selection.
Reset Pin (must be a valid C symbol)	IOPORT_PORT_06_PIN_00	Reset pin selection.
Slave Select Pin (SSL)(Must be a valid C symbol)	IOPORT_PORT_01_PIN_03	Slave select pin selection.
GT202 Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction)	5	GT202 driver task thread priority selection.
Callback	NULL	Callback selection.
Support NetX Packet Chaining	Enabled, Disabled Default: Enabled	Support NetX packet chaining selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different screen sizes. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are fairly intuitive and can usually be determined by

inspection of the associated Properties window from the SSP configurator.

Configuring the Wi-Fi Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules:

Configuration Settings for the SPI HAL Module on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first.
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	Clock synchronous operation	Select spi or clock syn mode operation.
Slave Select Polarity(SSL0)	Active Low	Select SSL0 signal polarity.
Slave Select Polarity(SSL1)	Active Low	Select SSL1 signal polarity.
Slave Select Polarity(SSL2)	Active Low	Select SSL2 signal polarity.
Slave Select Polarity(SSL3)	Active Low	Select SSL3 signal polarity.
Select Loopback1	Normal	Select loopback1.
Select Loopback2	Normal	Select loopback2.

Enable MOSI Idle State	Disable	Select MOSI idle fixed value and selection.
MOSI Idle State	MOSI Low	Select mosi idle fixed value and selection.
Enable Parity	Disable	Enable/disable parity.
Parity Mode	Parity Even	Select parity.
Select SSL(Slave Select)	SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3.
Select SSL Level After Transfer	SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep.
Clock Delay Enable	Clock Delay Disable	Clock delay enable selection.
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection.
SSL Negation Delay Enable	Negation Delay Disable	SSL negation delay enable selection.
Negation Delay Count	Negation Delay 1 RSPCK	Negation delay count selection.
Next Access Delay Enable	Next Access Delay Disable	Next access delay enable selection.
Next Access Delay Count	Next Access Delay 1 RSPCK	Next access delay count selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes Default: 2 Bytes	Transfer size selection. Note: For WiFi GT202 module, this property should be set to 1 Byte.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes Default: 2 Bytes	Transfer size selection. Note: For WiFi GT202 module, this property should be set to 1 Byte.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.

Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first.
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable.
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Bitrate Modulation Function enable or disable. Note: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

Configuration Settings for the Transfer Driver on Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ Driver on r_icu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling	Selection for trigger event mode
Digital Filtering	Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	TRUE	Determines if the interrupt is enabled immediately after initialization.
Callback	custom_hw_irq_isr	A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Interrupt priority selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Wi-Fi Framework Module Clock Configuration

The Wi-Fi Framework module uses the clocks required for the specific selections of the lower-level modules (such as the SPI).

Wi-Fi Framework Module Pin Configuration

The Wi-Fi Framework module uses input and output pins depending on the selections of the low-level modules (such as the SPI or the IRQ).

4.1.28.6 Using the Wi-Fi Framework Module in an Application

The following description is a high-level overview of some typical Wi-Fi use cases. A more detailed description and a working application project (which is too lengthy to include in this document), is available on the Renesas web site. Just search for the associated application note document number, r11an0226eu, in the top page search bar on www.Renesas.com. It is highly recommended that you use the application note to augment the summary descriptions found in this document.

Each of the Wi-Fi framework implementations are treated differently in a target application. The typical control flow for initialization, packet transmission using NetX/NetX Duo, packet reception using NetX/NetX Duo and using an on-chip networking stack are of particular interest. Example flow diagrams for some typical implementations of these functions are shown as follows:

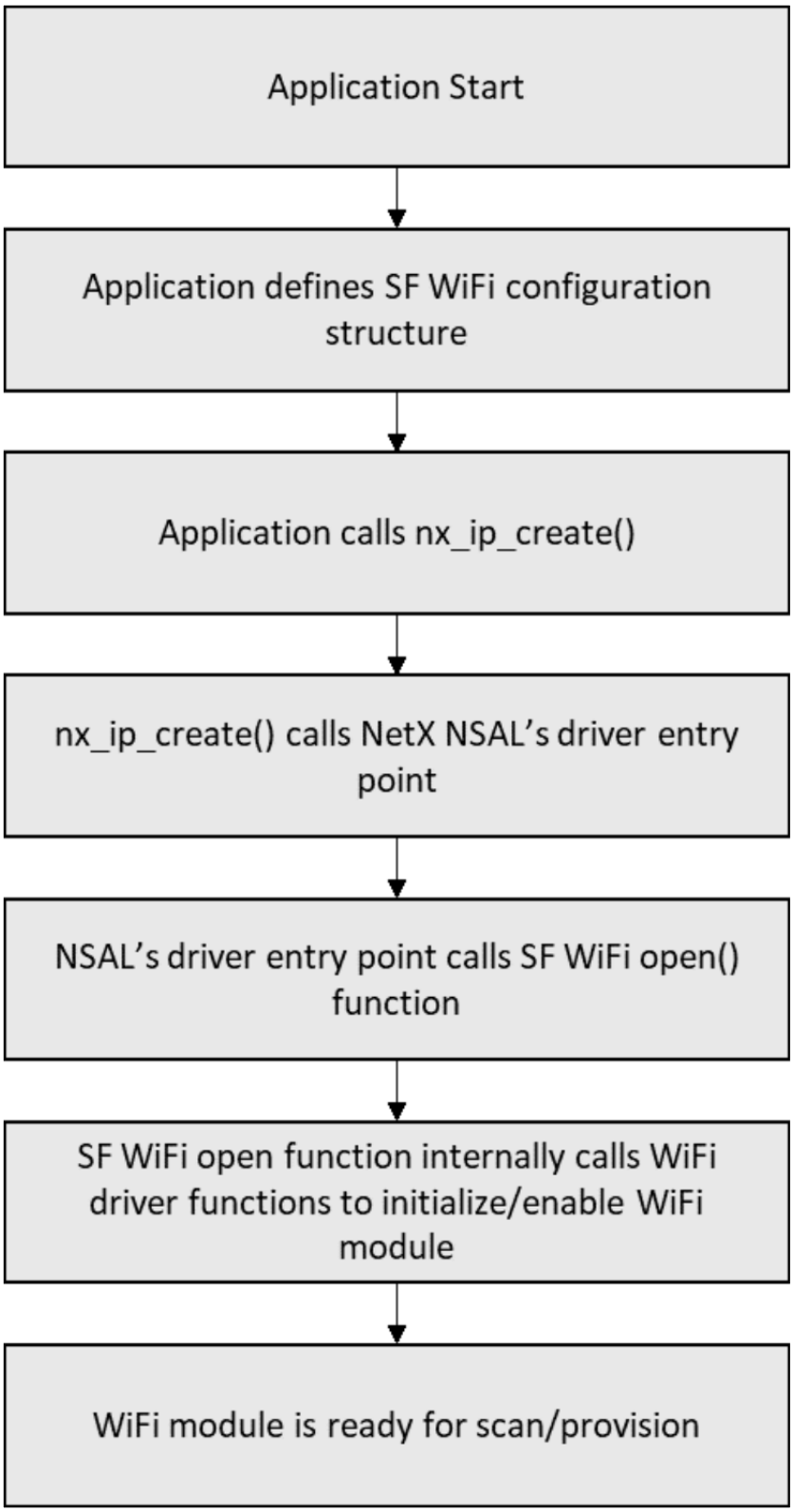


Figure 223: Application Control Flow using Wi-Fi Module Initialization

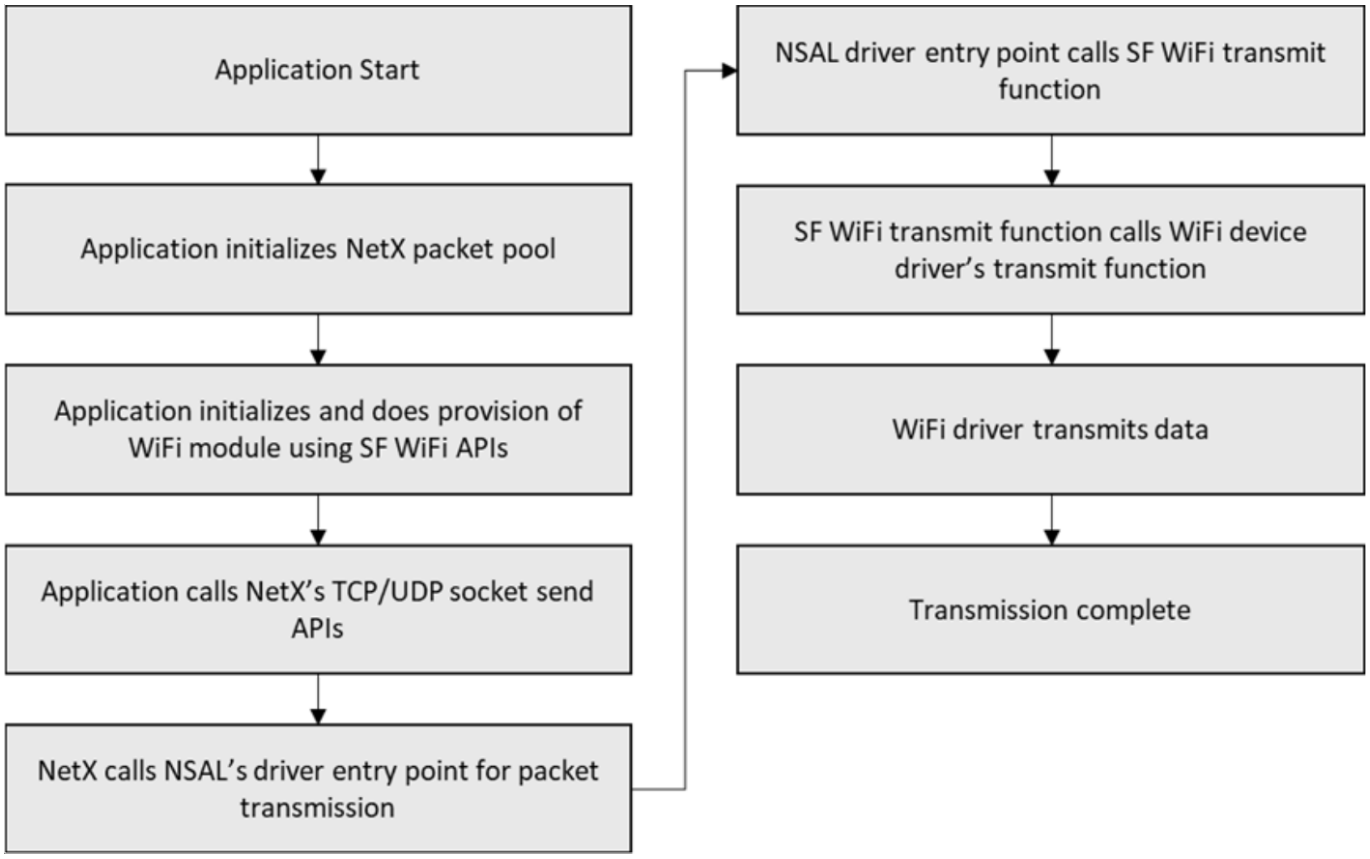


Figure 224: Application Control Flow Performing Packet Transmission using NetX

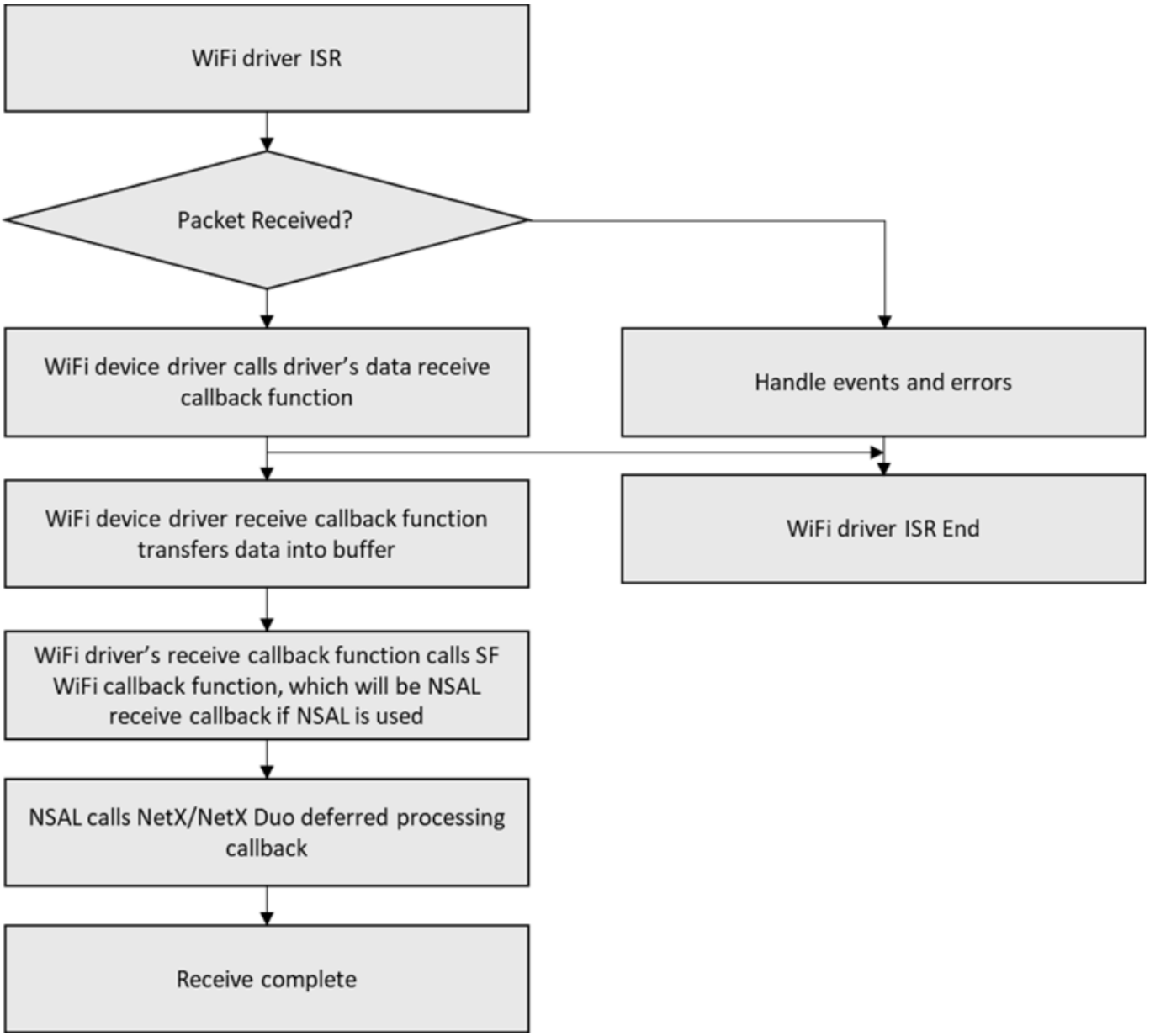


Figure 225: Application Control Flow Receiving Packets using NetX

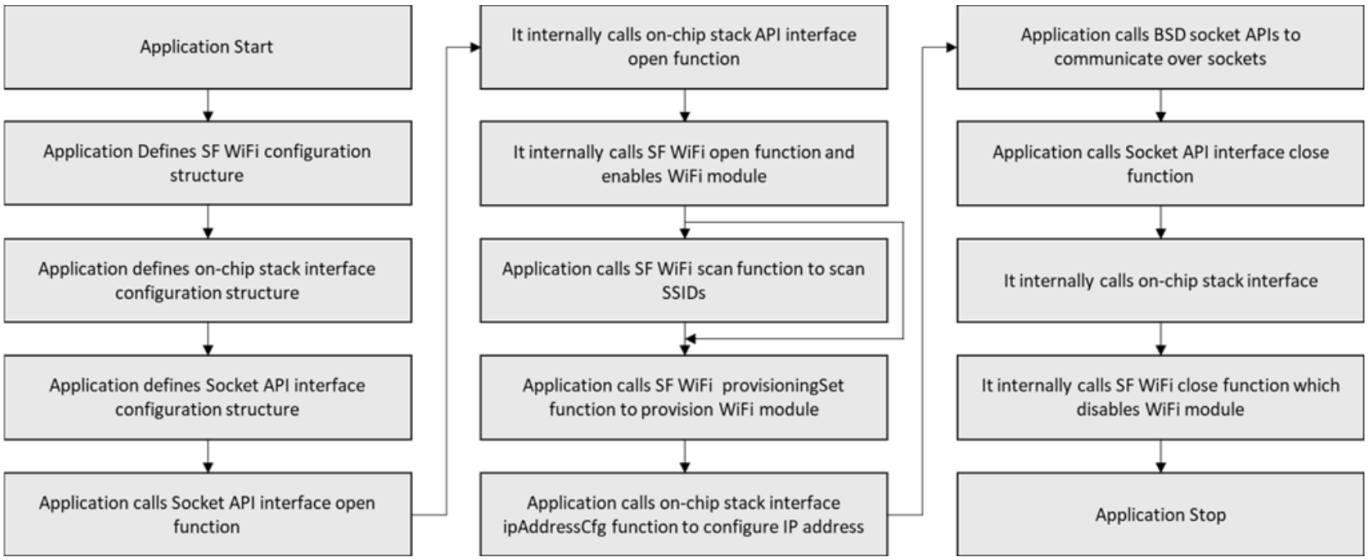


Figure 226: Application Control Flow using the On-Chip Networking Stack

Wi-Fi WPS Connection Methods

Flow diagrams for implementations of WPS connections for an application in both Station and AP modes for PIN and Button based connections are available in a Knowledge Base article here: https://en-support.renesas.com/knowledgeBase/18380440

4.1.29 Wi-Fi QCA4010 Framework

4.1.29.1 Wi-Fi QCA4010 Framework Introduction

The SX-ULPGN is a low-power, compact IEEE 802.11b/g/n 2.4GHz 1x1 Wireless LAN module equipped with the Qualcomm® QCA4010 Wireless SOC.

The Wi-Fi QCA4010 framework provides a high-level API for configuring and provisioning Silex QCA4010 ULPGN module as well as perform TCP and UDP data transfers with on-chip networking capability.

Wi-Fi QCA4010 Framework Module Features

- Provides high-level APIs to configure and provision a SX-ULPGN Wi-Fi module.
- Provides three different implementations for:
    - A Wi-Fi device driver stack using the sf_wifi_qca4010 framework.
    - An on-chip stack using the sf_wifi_qca4010_onchip_stack framework.
    - A socket stack using the sf_wifi_qca4010_socket framework.
- Provides a socket interface to create socket-based applications with the On-chip TCP/UDP.

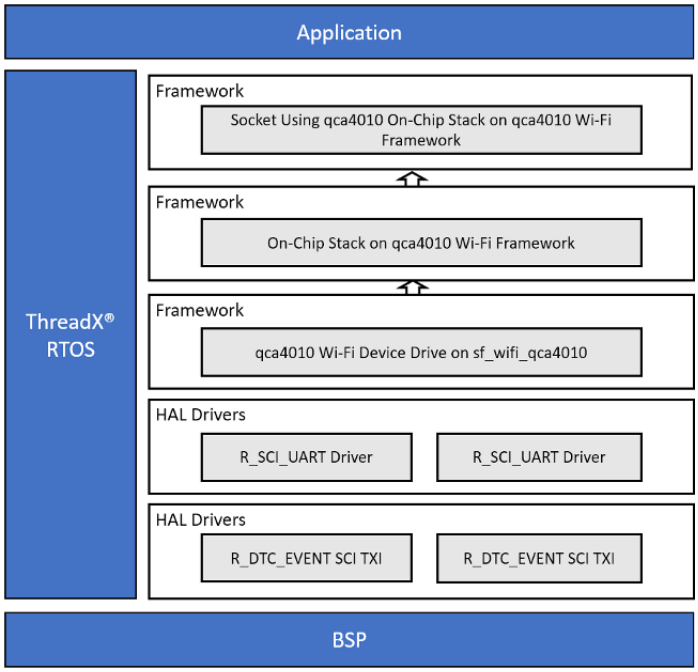


Figure 227: Wi-Fi QCA4010 Framework Module Block Diagram

Note

The On-Chip Stack on qca4010 Wi-Fi Framework, qca4010 Wi-Fi Device Driver on sf_wifi_qca4010 can be used as a lower-level implementation of the Socket Framework or on its own.

4.1.29.2 SF WIFI QCA4010 Framework APIs Overview

The QCA4010 Wi-Fi Framework provides APIs for features such as scanning, provisioning with access points, initialization of access point with different security types (WPA2, WPA, WEP, Open), enabling/disabling DHCP server in AP mode, sending AT commands directly to wifi module and obtain the response, creation of TCP and UDP sockets, TCP and UDP data transfers.

Complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Wi-Fi QCA4010 APIs

Wi-Fi QCA4010 APIs

API Name	Example API call and description
open	g_sf_wifi_qca4010.p_api->open(g_sf_wifi_qca4010.p_ctrl, g_sf_wifi_qca4010.p_cfg); Open lower level driver of Wi-Fi framework.
close	g_sf_wifi_qca4010.p_api->close(g_sf_wifi_qca4010.p_ctrl); Close lower level driver of Wi-Fi framework.

provisioningSet	g_sf_wifi_qca4010.p_api->provisioningSet(g_sf_wifi_qca4010.p_ctrl, &provisioning_info); Provision/Connect to Access Point or Initiate an access point with information provided by user such as mode (AP/station), channel, SSID, security type (WPA2/WPA/WEP/Open), WEP key index, encryption type(TKIP/CCMP), passphrase/key.
wifiStatusGet	g_sf_wifi_qca4010.p_api->wifiStatusGet(g_sf_wifi_qca4010.p_ctrl, &wifi_status); Get the status of connected or Initiated Access Point.
scan	g_sf_wifi_qca4010.p_api->scan(g_sf_wifi_qca4010.p_ctrl, scan_result, scan_count); San for available access points.
versionGet	g_sf_wifi_qca4010.p_api->versionGet(&version); Get the version based on compile time macros.
CommandSend	g_sf_wifi_qca4010.p_api->CommandSend(g_sf_wifi_qca4010.p_ctrl, &cmd, &resp, timeout); Send AT command directly to Wifi module and obtain the response.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.
For provisioningSet API, user has to provide information about the access point such as SSID, security type, Key(Password), encryption type, channel and mode. SSID and passphrase cannot contain commas. This is a current limitation of Silex module firmware.

Wi-Fi QCA4010 On chip Stack APIs

Wi-Fi QCA4010 On chip Stack APIs

API Name	Example API call and description
open	g_sf_wifi_qca4010_onchip_stack.p_api->open(g_sf_wifi_qca4010_onchip_stack.p_ctrl, g_sf_wifi_qca4010_onchip_stack.p_cfg); Open on-chip stack layer of Wi-Fi framework.
close	g_sf_wifi_qca4010_onchip_stack.p_api->close(g_sf_wifi_qca4010_onchip_stack.p_ctrl); Close on-chip stack layer of Wi-Fi framework.
ipAddressCfg	g_sf_wifi_qca4010_onchip_stack.p_api->ipAddressCfg(g_sf_wifi_qca4010_onchip_stack.p_ctrl, &ip_cfg); Set static IP or Enable DHCP based on user input.

ping	g_sf_wifi_qca4010_onchip_stack.p_api->ping(g_sf_wifi_qca4010_onchip_stack.p_ctrl, &ping_ip,0, 0); Ping the IP address provided by the user.
versionGet	g_sf_wifi_qca4010_onchip_stack.p_api->versionGet(&version); Get the version based on compile time macros.
dhcpServerStart	g_sf_wifi_qca4010_onchip_stack.p_api->dhcpServerStart (g_sf_wifi_qca4010_onchip_stack.p_ctrl, &start_ip, &end_ip) Start DHCP server in AP mode.
dhcpServerStop	g_sf_wifi_qca4010_onchip_stack.p_api->dhcpServerStop(g_sf_wifi_qca4010_onchip_stack.p_ctrl, &start_ip, &end_ip) Stop DHCP server in AP mode.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.
DHCP server has to be used in AP mode and not recommended to be used with station mode. User must set a fixed/static IP address to the AP using the API 'ipAddressCfg' and then start DHCP server.

Wi-Fi QCA4010 Socket APIs

Wi-Fi QCA4010 Socket APIs

API Name	Example API call and description
open	g_sf_wifi_qca4010_socket.p_api->open(g_sf_wifi_qca4010_socket.p_ctrl, g_sf_wifi_qca4010_socket.p_cfg); Open Socket layer of Wi-Fi Framework.
close	g_sf_wifi_qca4010_socket.p_api->close(g_sf_wifi_qca4010_socket.p_ctrl); Close Socket layer of Wi-Fi Framework.
versionGet	g_sf_wifi_qca4010_socket.p_api->versionGet(&version); Get the version based on compile time macros.
socketCreate	g_sf_wifi_qca4010_socket.p_api->socketCreate(g_sf_wifi_qca4010_socket.p_ctrl, 0, SOCK_STREAM, AF_INET); Create TCP/UDP socket
socketConnect	g_sf_wifi_qca4010_socket.p_api->socketConnect (g_sf_wifi_qca4010_socket.p_ctrl, 0, (structsockaddr*)&sock_addr,addrlen); Connect TCP/UDP client to Server.

socketDisconnect	g_sf_wifi_qca4010_socket.p_api->socketDisconnect(g_sf_wifi_qca4010_socket.p_ctrl, 0); Disconnect socket connection and close the network.
socketSend	g_sf_wifi_qca4010_socket.p_api->socketSend(g_sf_wifi_qca4010_socket.p_ctrl, 0, (uint8_t*)&data, length, 1000); Send data from TCP/UDP socket.
socketRecv	g_sf_wifi_qca4010_socket.p_api->socketRecv(g_sf_wifi_qca4010_socket.p_ctrl, 0, (uint8_t*)&data, 1024, 1000); Receive datafromTCP/UDPserver.
socketStatusGet	g_sf_wifi_qca4010_socket.p_api->socketStatusGet(g_sf_wifi_qca4010_socket.p_ctrl, 0, (uint32_t*)&socket_status); Check the status of socket created.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.
For socketCreate API, user has to pass parameter to select TCP or UDP socket, family (IPV4). The parameter socket number should be '0' for single socket and while creating multiple sockets, socket number of first socket has to be '0' and increments by 1 for the next socket.
For socketConnect API, user has to pass Server IP address, family (IPV4), Port number

Wi-Fi QCA4010 Framework Status Return Values

Wi-Fi QCA4010 Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_WIFI_FAILED	API call not successful.
SSP_ERR_ASSERTION	An input pointer is Null
SSP_ERR_ALREADY_OPEN	Unit is already opened
SSP_ERR_WIFI_INIT_FAILED	Driver initialization failed.
SSP_ERR_IN_USE	Module is already in use
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_INTERNAL	Internal error occurred
SSP_ERR_TIMEOUT	Timeout error
SSP_ERR_UNSUPPORTED	Functionality (IPV6) not supported

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the

associated module for a definition of all relevant status return values.

4.1.29.3 SF_WIFI_QCA4010 Framework Module Operational Overview

The Wi-Fi framework provides a high-level interface for the application to configure the Wi-Fi module, provision the Wi-Fi module in station or AP mode with different security types and perform TCP/UDP data transfers in server and client mode.

The following figure provides an overview of the QCA4010 Wi-Fi framework layered architecture.

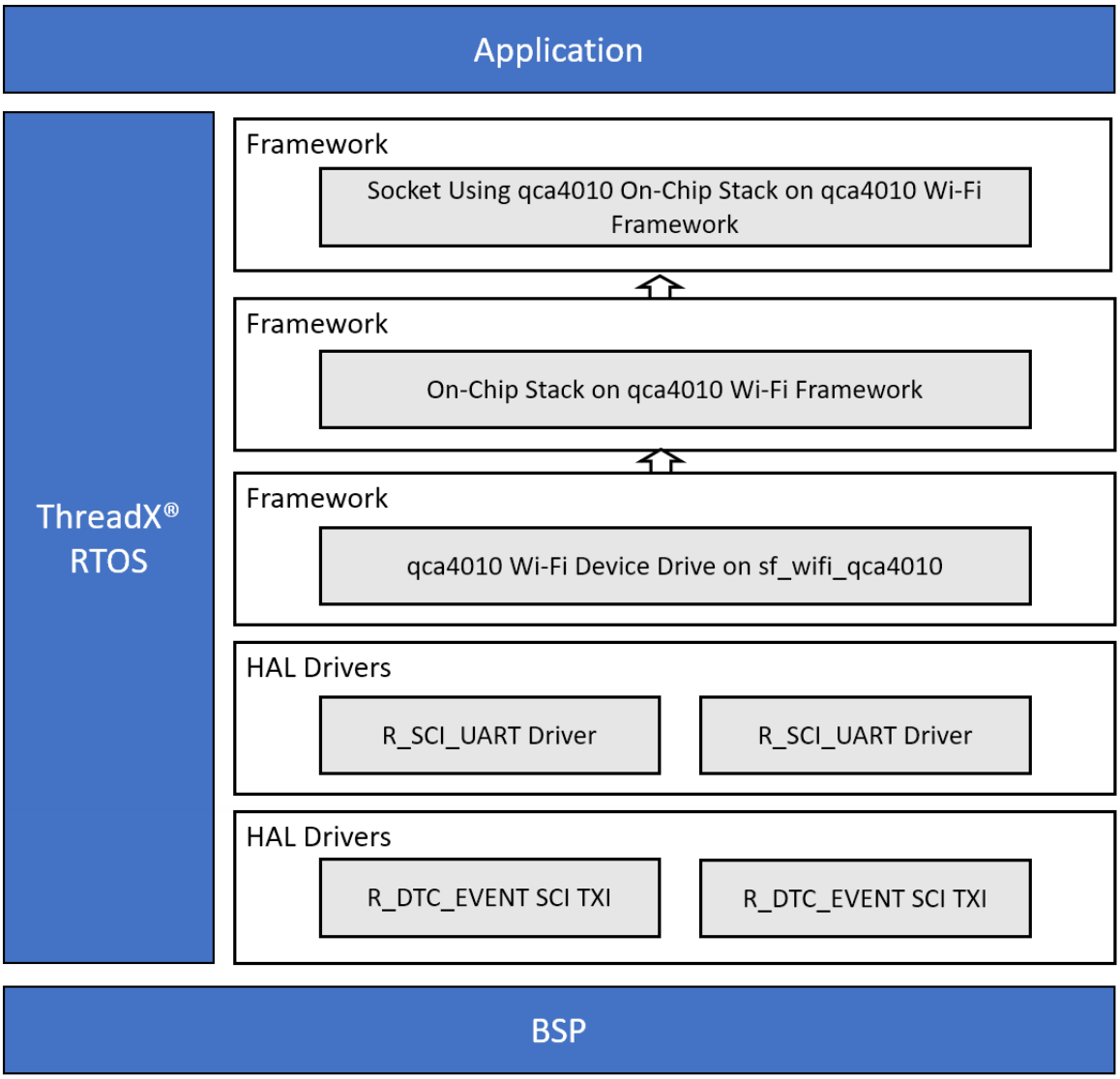


Figure 228: Wi-Fi QCA4010 Framework Module Stack Implementation

- Socket stack using the sf_wifi_qca4010_socket framework provides APIs to create, connect TCP/UDP sockets and perform data send and receive operations in client/server mode.
- On-chip stack using the sf_wifi_qca4010_onchip_stack framework provides APIs to configure static IP or enable DHCP, ping, Enable/Disable DHCP server.
- Wi-Fi device driver stack using the sf_wifi_qca4010 framework provides APIs to initialize Wifi module, provision the module in station or AP mode, scan for available access points, get

the status of connected or initiated access point, send AT commands directly to Wifi module.

Note

The On-Chip Stack on qca4010 Wi-Fi Framework, qca4010 Wi-Fi Device Driver on sf_wifi_qca4010 can be used as a lower-level implementation of the Socket Framework or on its own.

SF_WIFI_QCA4010 Framework Module Important Operational notes and Limitations

SF_WIFI_QCA4010 Framework Module Operational Notes

Operational Overview

SF_WIFI_QCA4010 Framework is used to:

1. Scan available access points.
2. Provision/connect to the desired access point with the information provided by user such as SSID, security type, Key, encryption type, WEP key index, channel, Mode.
3. Initiate an access point with different security types (WPA2/WPA/WEP/Open).
4. Enable/Disable DHCP server in AP mode.
5. Obtain the status of connected access point or initiated AP such as SSID, Phy Mode, MAC Address, Mode, Channel.
6. Set a static IP or enable DHCP and obtain information such as IP Address, NetMask and Gateway of the Wifi Module.
7. Ping the server IP address.
8. Send custom AT commands directly to wifi module and obtain the response.
9. Create single/multiple TCP/UDP sockets.
10. Connect TCP/UDP client to the server and vice-versa.
11. Obtain the current socket status.
12. Send or receive data from TCP/UDP server/client.
13. Disconnect the socket and close the connection.

Note

Wi-Fi framework supports 1 or 2 UARTs for interfacing with the SX-ULPGN module. The second UART is considered optional for single socket and is mandatory for multiple socket operation.
The number of sockets that can be created is based on the memory availability of board.
*SX-ULPGN supports WEP security with WEP key as **10 or 26 digit hexadecimal string**.*

SF_WIFI_QCA4010 Framework Module limitations

Wi-Fi QCA4010 Framework limitations

1. The Wi-Fi Framework does not support the Synergy S1 board series due to memory constraints.

2. Only IPV4 protocol version is supported as majority of the Silex AT commands does not support IPV6.

1. The SX-ULPGN chip does not support ethernet pass through. Hence NetX stack and related services are not supported. Only on-chip stack and related APIs are supported.

4. WPS (PUSH/PIN) method of station provisioning is not supported.

SX-ULPGN Hardware limitations:

1. SX-ULPGN chip doesn't support ethernet pass through.

2. WPA 2 Enterprise is not supported by SX-ULPGN.

3. SX-ULPGN supports only 2.4GHz frequency band, 5GHz band is not supported.

4.1.29.4 Including the SF_WIFI_QCA4010 Framework in an Application

This section describes how to include SF_WIFI_QCA4010 framework in an application using the SSP configurator.

Note

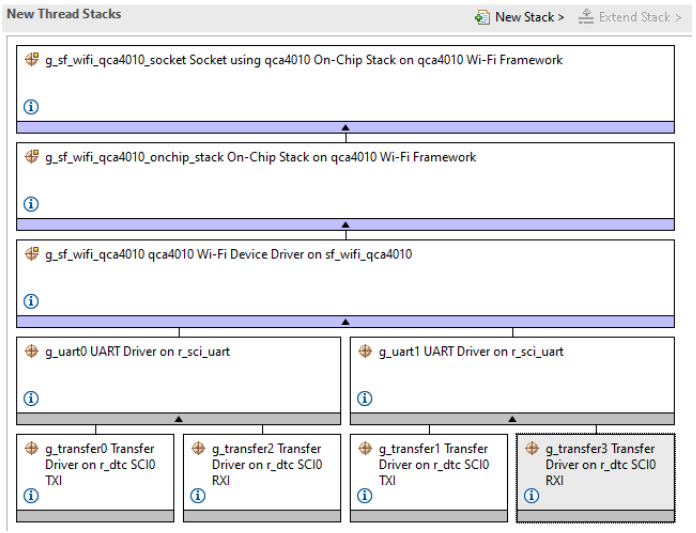
This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the sf_wifi_qca4010 Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Wi-Fi QCA4010 Framework module is g_sf_wifi_qca4010_socket. This name can be changed in the associated Properties window.)

Wi-Fi QCA4010 Framework Module Selection Sequence

Resource	ISDE tab	Stack selection sequence
g_sf_wifi_qca4010_socket Socket using qca4010 On-Chip Stack on qca4010 Wi-Fi Framework	Threads	New Stack> Framework> Networking> Wi-Fi>Socket using qca4010 On-Chip Stack on qca4010 Wi-Fi Framework
g_sf_wifi_qca4010_onchip_stack On-Chip Stack on qca4010 Wi-Fi Framework	Threads	New Stack> Framework> Networking> Wi-Fi>On-Chip Stack on qca4010 Wi-Framework
g_sf_wifi_qca4010 qca4010 Wi-Fi Device Driver on sf_wifi_qca4010	Threads	New Stack> Framework> Networking> Wi-Fi>qca4010 Wi-Fi Device Driver on sf_wifi_qca4010

When the SF_WIFI_QCA4010 Framework is added to the thread stack as shown in the above figure, the configurator automatically adds any needed lower-level module. Any modules needing additional configuration information have the box text highlighted in Red (Here additional UART driver has to be added by the user for multiple socket operations). Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.



4.1.29.5 Configuring the Wi-Fi QCA4010 Framework

The Wi-Fi QCA4010 must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for g_sf_socket0 on qca4010 Wi-Fi Framework

ISDE Property	Value	Description
Parameter checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Number of supported socket instances	1 – 12 Default :1	Number of sockets to be created.
Name	g_sf_socket0	Name of upper level framework layer.
Name of generated initialization function	sf_socket_init0	Name of auto generated function.
Auto Initialization	Enable, Disable Default: Enable	Selects if code for auto-initialization to be included in the build.

Configuration Settings for g_sf_wifi_onchip_stack0 on Wi-Fi Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_wifi_onchip_stack0	Name of on-chip stack layer.

Configuration Settings for g_sf_wifi_qca40100 on qca4010 Wi-Fi Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
AT command Retry count	5 – 255 Default: 10	Number of retries of issuing AT command in case of failure in response.
Hardware Mode	802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode of Wi-Fi Module.
Reset Pin	Default: IOPORT_PORT_10_PIN_05	Wi-Fi module reset IO pin selection.
Queue size in bytes	256 - 512 Default: 256	UART reception queue size.
Response Buffer size	256 - 4096 Default: 2048	Response Buffer size selection.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuring the Wi-Fi QCA4010 Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the lower-level modules:

Configuration Settings for r_sci_uart driver

ISDE Property	Value	Description

External RTS Operation	Disable, Enable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality, set this configuration parameter to Enable and specify the configuration Name of UART callback function for the RTS external pin control.
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to Disable reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting this configuration to Disable reduces code size because the portion of code for UART transmission is not compiled. However, you can only set this configuration to Disable if no other SCI channels which work as UART ports are transmitting.
Parameter checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0-9 Default : 0	SCI channel number.
Baud rate	Default : 9600	Baud rate selection.
Data bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

CTS/RTS selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select CTS for this configuration parameter and enable the configuration External RTS Operation specifying the configuration Name of UART callback function for the RTS external pin control.
Name of UART callback function	sf_wifi_qca4010_serial_uart_callback	UART callback to receive data/AT command response.
Name of UART callback function for the RTS external pin control to be defined by the user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set Falling Edge to this configuration.
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For details, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection.

Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Error interrupt priority selection.
Baud rate Percent Error	Value must be greater than 0.0 and less than 15.0 Default; 2.0	Maximum baud rate percent error allowed in order for the module to function
UART Communication Mode	RS232, RS485 Default: RS232	UART communication mode selection, usually it is RS232 mode
UART RS485 Communication Mode	Half Duplex, Full Duplex Default: Half duplex	UART RS485 communication mode selection as half duplex or full duplex
RS485 DE Port	00 to 11 Default: 09	Select the port number of Driver Enable Pin
RS485 DE Pin	00 to 15 Default: 14	Select the pin number of Driver Enable Pin

Note

Additional UART is mandatory for multiple socket operation and is optional for single socket operation.

Configuration of second UART driver instance is same as first UART instance which is described above.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE	Property	Value Description
Parameter Checking	Enabled, Disabled Default: Disabled	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Set start mode
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section setting
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	True, False Default: True	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection

Addition of DTC driver for reception is not recommended.

Note

The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

SF_WIFI_QCA4010 Framework Module Clock configuration

SF_WIFI_QCA4010 Framework has no specific clock configuration requirements.

SF_WIFI_QCA4010 Framework Module pin configuration

The SF_WIFI_QCA4010 Framework module uses input and output pins of the lower level UART driver.

4.1.29.6 Using the Wi-Fi QCA4010 Framework Module in an Application

Wi-Fi framework can be used to initialize access point (AP mode) or connect client to AP/router (station mode)

The typical steps of using the Wi-Fi Framework in an application for station mode are:

1. Initialize the Wi-Fi module using sf_wifi_qca4010_socket_api_t::open API.
2. Scan for the available access points using sf_wifi_qca4010_api_t::scan API.
3. Connect to the desired access point using sf_wifi_qca4010_api_t::provisioningSet API. User has to provide the information such as SSID, security type (WPA2/WPA/WEP/Open), Key, Mode, Channel, encryption type, WEP key index of the access point to be connected.
4. After connection is successful, obtain the status of the connected AP using sf_wifi_qca4010_api_t::wifiStatusGet API.
5. Configure IP address(static or Dynamic) using sf_wifi_qca4010_onchip_stack_api_t::ipAddressCfg API.
6. Ping the server IP using sf_wifi_qca4010_onchip_stack_api_t::ping API.
7. After ping is successful, create and connect single/multiple TCP/UDP sockets in client or server mode using sf_wifi_qca4010_socket_api_t::socketCreate and sf_wifi_qca4010_socket_api_t::socketConnect API's.
8. Send data from TCP/UDP client to server using sf_wifi_qca4010_socket_api_t::socketSend API.
9. Receive data from TCP/UDP server using sf_wifi_qca4010_socket_api_t::socketRecv API.
10. Disconnect single/multiple sockets using sf_wifi_qca4010_socket_api_t::socketDisconnect API to close the network connection.
11. De-Initialize the Wi-Fi module using sf_wifi_qca4010_socket_api_t::close API.

The typical steps of using the Wi-Fi Framework in an application for AP mode are:

1. Initialize the Wi-Fi module using sf_wifi_qca4010_socket_api_t::open API.
2. Initialize/provision access point using sf_wifi_qca4010_api_t::provisioningSet API. User has to provide the information such as SSID, security type (WPA2/WPA/WEP/Open), Key, Mode, Channel, encryption type, WEP key index to the access point.
3. Set a static IP address using sf_wifi_qca4010_onchip_stack_api_t::ipAddressCfg API.
4. Enable DHCP server using sf_wifi_qca4010_onchip_stack_api_t::dhcpServerStart API by specifying start and end IP address.
5. Once the above mentioned steps are successful, client can connect to the initiated access point and obtain the IP address in between the range set by DHCP server of AP.
6. After connection is successful, user can perform operations such as ping, TCP/UDP data transfers.

Note

*SX-ULPGN supports WEP security with either **10 or 26 digit hexadecimal string**.*

4.2 HAL Layer

I2S Driver

GPT Input Capture on r_gpt Driver

I/O Port Driver

Watchdog Driver on r_iwdt

JPEG Decode Driver

JPEG Encode Driver

Key Matrix Driver

Low Power Modes Driver on r_lpmv2

Low Voltage Detection Driver

OPAMP Driver

PDC Driver

PTP Driver on r_ptp

PTPEDMAC Driver on r_ptpedmac

QSPI Driver

RTC Driver

SCE Crypto Driver

SDADC Driver

SD/MMC Driver and SDIO Driver

Segment LCD Driver

SCI SPI Driver

SPI Driver

UART Driver

Watchdog Driver

4.2.1 Analog Connection Driver on r_analog_connect

4.2.1.1 Analog Connection HAL Module Introduction

The analog connection HAL module implements the analog connect API on r_analog_connect to simplify the connection of the analog components that comprise the analog front end on select Synergy MCUs. Previously these connections needed to be made using low-level register-based configuration instead of using the Synergy configurator. The analog connection module supports the OPAMP (operational amplifier), ACMPHS (high speed analog comparator), and ACMPLP (low power analog comparator) peripherals, and their allowed interconnections, available on the Synergy microcontroller hardware.

Analog Connection HAL Module Features

- Simplifies the interconnection of internal analog connections
- Selection of interconnects is limited to those available to the target MCU, simplifying the configuration process and eliminating common configuration errors
- Uses the time saving and intuitive SSP configuration GUI

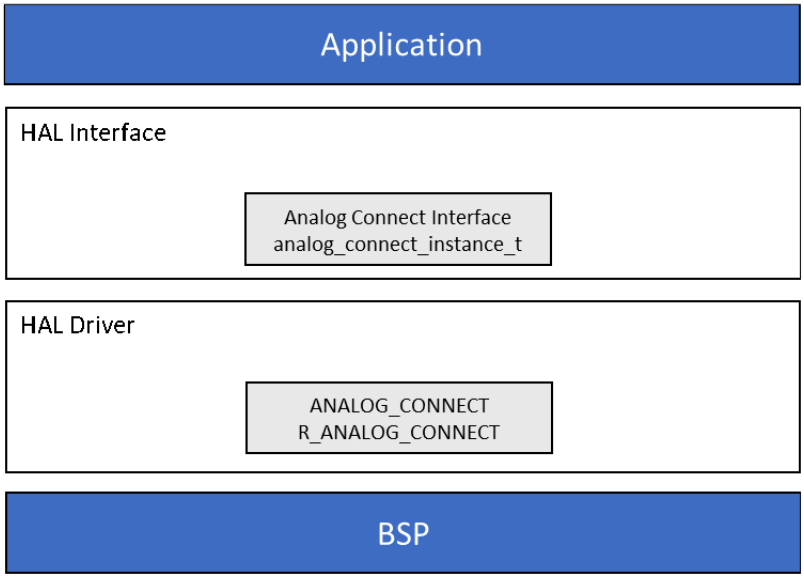


Figure 229: Analog Connection HAL Module Block Diagram

4.2.1.2 Analog Connection HAL Module APIs Overview

The Analog Connection HAL module defines API functions for making single or multiple connections. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Analog Connection HAL Module API Summary

Function Name	Example API Call and Description
init	g_analog_connect.p_api ->init(g_analog_connect.p_cfg); Initialize the analog connection module.

connect	g_analog_connect.p_api ->connect(ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013); Make connection between specified analog module input and output using a connection option enum. Enum definitions are explained in the operational notes later in this document.
connectMultiple	g_analog_connect.p_api->connectMultiple(&connection_table); Make all connection between specified analog module input and output as specified in the connection table analog_connect_table_t.
versionGet	g_analog_connect.p_api ->versionGet (&version); Get the version information of the underlying driver.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Analog Connection HAL Module Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully
SSP_ERR_ASSERTION	Data table pointer is NULL or size is 0

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.1.3 Analog Connection HAL Module Operational Overview

The analog connection HAL module controls the internal analog connections for the OPAMP, ACMPLP, and ACMPHS peripherals on a Synergy microcontroller. It directly controls the hardware without using any RTOS elements and provides convenient APIs to simplify development.

Analog Connection HAL Module Important Operational Notes and Limitations

Analog Connection HAL Module Operational Notes

The following operational notes describe the available connections, the enumeration format used to describe connections, general notes and module specific notes on operations important when using the analog connect module in an application.

Available Connections

The list of connection options is unique for each MCU. This list is found in the analog_connect_t enumeration in synergy\ssp\src\bsp\mcu\<mcu_name>\bsp_analog.h, where <mcu_name> is the name of the MCU series, for example S7G2.

You can find the descriptions of the analog_connect_t enumeration for each MCU as follows:

- analog_connect_t for S124
- analog_connect_t for S128
- analog_connect_t for S1JA
- analog_connect_t for S3A1
- analog_connect_t for S3D3
- analog_connect_t for S3A6
- analog_connect_t for S3A7
- analog_connect_t for S5D5
- analog_connect_t for S5D9
- analog_connect_t for S7G2

The connection enum format is described below:

- ANALOG_CONNECT_<NODE>_TO_<NODE> is used to make a connection
  - <NODE> can be further divided into <PERIPHERAL><CHANNEL>_<SIGNAL>
  - <PERIPHERAL> is the peripheral name: ACMPHS, ACMPLP, or OPAMP
  - <CHANNEL> is the channel number
  - <SIGNAL> is the signal name. The signal name corresponds to the signal name in the hardware manual
- ANALOG_CONNECT_<NODE>_BREAK is used to break all connections to a node. This option is currently available for OPAMP connections only.

The connection options are grouped by peripheral, channel, and signal in the analog_connect_t enumeration. All signals should be configured for each peripheral/channel combination used. For example, if an application uses ACMPHS channel 0, then exactly one connection must be made for each node that begins with ACMPHS0. Exactly one connection must be made for ACMPHS0_IVREF and ACMPHS0_IVCMP. Likewise, if an application uses OPAMP channel 0, then a connection must be made for OPAMP0_AMPP, OPAMP0_AMPM, and OPAMP0_AMPO.

General Notes Regarding Connections

The analog connect module makes internal connections only. If an application interfaces to analog signals external to the MCU, it will also require pin configurations on the **Pins** tab in the Synergy Configuration Tool.

Calling analog_connect_api_t::connect or analog_connect_api_t::connectMultiple overwrites any existing connection and replaces it with the new connection.

The APIs in this module have no prerequisite. They can be called before or after the associated driver's comparator_api_t::open() or opamp_api_t::open(). If the analog_connect_api_t::connect or analog_connect_api_t::connectMultiple APIs are called before the open() of the associated driver, the module stop bit for the associated peripheral channel is cleared.

The analog_connect_api_t::connectMultiple API is most efficient when connections for the same peripheral/channel combination are grouped together in the list of connections.

Connection Aliases

Not all connection aliases are listed in the analog_connect_t enumeration. For example, take the S1JA connection to connect the IVCMP input of the ACMPHS comparator to P013 (ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013). P013 can be used as the DAC8 channel 0 output. If P013 is used as the DAC8 channel 0 output, then making this connection connects the DAC8 channel 0 output to the ACMPHS0 IVCMP input. Refer to the Synergy Configuration tool **Pins** tab or the hardware manual I/O Ports chapter, "Peripheral Select Settings for each Product" section for pin aliasing options.

Multiple Connection Table

The table used to make multiple connections includes the number of connections followed by the list of connections. The multiple connection table structure follows the below definition:

```
typedef struct st_analog_connect_table_t
{
uint32_tnumber_of_connections;
analog_connect_tconst*p_connection_table;
} analog_connect_table_t;
```

ACMPLP Connections Operational Notes

ACMPLP connections should only be made before comparator_api_t::outputEnable() is called. To modify connections, call comparator_api_t::close() first.

On some MCUs, the ACMPLP peripheral has 2 internal nodes that can be configured. They are called ACMPLP0_IVREF0 and ACMPLP1_IVREF1. These nodes must be configured exactly once if they are used. Two enums must be used to make connections involving these nodes. For example, to connect ACMPLP channel 0 to the output of DAC8 channel 0, the following two connections must be made:

- To connect the - input of the comparator (IVREF) to the internal node IVREF0: ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0
- To connect the internal node IVREF0 to the output of DAC8 channel 0: ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA

The node ACMPLP0_IVREF0 can be connected to the IVREF signal of either channel of the ACMPLP (ACMPLP0_IVREF or ACMPLP1_IVREF). The user should be aware when modifying this node that it may affect the setting of the other channel.

ACMPHS Connections Operational Notes

On MCUs that have an ADC PGA, the associated ADC PGA must be configured or bypassed to use IVCMP2 as the IVCMP input to the comparator. If the associated ADC unit is open, the PGA is bypassed. To enable the ADC PGA or bypass the PGA without opening the ADC driver, follow the instructions in the Usage Notes section of the ADC chapter of the hardware manual. This chapter typically starts with "Available Functions and Register Settings of AN000 to AN002".

ACMPHS connections can be reconfigured at runtime. If an ACMPHS connection is reconfigured after comparator_api_t::outputEnable() is called, the comparator will be disabled before reconfiguring the connection. If the comparator was already enabled, it will be re-enabled after a stabilization wait time of 48 ICLK.

OPAMP Connections Operational Notes

OPAMP connections are only supported on MCUs that have AMPnPS, AMPnMS, and AMPnOS registers. OPAMP connections are only listed in the analog_connect_t enumeration if they exist.

OPAMP connections require a change pump to be enabled if AVCC0 < 2.7 V. Make sure the setting

for MCU Analog Power Supply AVCC0 (mV) on the **BSP** tab of the Synergy Configuration tool is correct if OPAMP connections are used. If AVCC0 < 2.7 V, the MOCO must also be operating to use the charge pump. When using the charge pump for the amplifier:

- Turn on no more than a total of 5 connections for OPAMP0.
- Turn on no more than a total of 5 connections for OPAMP1.
- Turn on no more than a total of 2 connections for OPAMP2.

OPAMP connections can be reconfigured at runtime.

OPAMP connections can be combined by OR'ing connections for the same node. Connection enumerations that start with the same value up to the '_TO_' part of the enum name (ANALOG_CONNECT_<NODE>_TO_) can be OR'ed together. See the paragraph above regarding charge pump operation for limits on the total number of connections that can be made when the charge pump is operating. Any number of connections can be made if AVCC0 >= 2.7 V.

Warning

NEVER combine connections for different nodes. The analog connect module will not behave as expected if connections for different nodes are combined.

An illustration of combining connections is shown in the code snippet below:

```
/* Connect OPAMP0 AMP+ to the output of DAC12 channel 0 and P013. These connections
can be combined since both start with ANALOG_CONNECT_OPAMP0_AMPP_TO_. */
g_analog_connect.p_api->connect(ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA |
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013);
/* Connect OPAMP0 AMP+ to just P013 (this disconnects the DAC12 channel 0 input). */
g_analog_connect.p_api->connect(ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013);
```

If a connection must be broken before a new connection is made, use the _BREAK enumeration. This is illustrated by the code snippet that follows.

```
/* Connect OPAMP1 AMP+ to the output of DAC12 channel 0 and P002. */
g_analog_connect.p_api->connect(ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002);
/* Break all connections to OPAMP1 AMP+ (in this case the connection to P002), then
establish a new connection to P003. */
g_analog_connect.p_api->connect(ANALOG_CONNECT_OPAMP1_AMPP_BREAK);
g_analog_connect.p_api->connect(ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003);
```

Analog Connection HAL Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.1.4 Including the Analog Connection HAL Module in an Application

This section describes how to include the analog connection HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the analog connection HAL module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the analog connect module is g_analog_connect0. This name can be changed in the associated Properties window.)

Analog Connection HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_analog_connect0 Analog Connection Driver on r_analog_connect	Threads	**New Stack> Driver> Analog> Analog ConnectionDriver on r_analog_connect<MCU>**

When the Analog Connection Driver on r_analog_connect is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

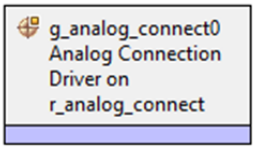


Figure 230: Analog Connection HAL Module Stack

4.2.1.5 Configuring the Analog Connection HAL Module

The Analog Connection HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Analog Connection HAL Module on r_analog_connect

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_analog_connect0	Module name.
Connection Table Name	g_analog_connect0_table	This connection table name is passed to the connectMultiple API function to make all configured connections.
ACMPHS0 IVCMP	No Connection, PORT5 P500, PORT0 P013, PORT1 P100 Default: No Connection	Select the connection for ACMPHS0 IVCMP.
ACMHS0 IVREF	No Connection, PORT5 P501, PORT0 P014, PORT1 P101, DAC80 DA, DAC120 DA, Analog0 VREF Default: No Connection	Select the connection for ACMPHS0 IVREF.
ACMPL0 IVREF0	No Connection, PORT1 P109, DAC80 DA Default: No Connection	Select the connection for ACMPL0 IVREF0.
ACMPLP1 IVREF1	No Connection, PORT1 P110, DAC81 DA Default: No Connection	Select the connection for ACMPLP1 IVREF1.
ACMPLP0 IVCMP	No Connection, PORT4 P400, OPAMP0 AMPO Default: No Connection	Select the connection for ACMPLP0 IVCMP.
ACMPLP0 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0 Default: No Connection	Select the connection for ACMPLP0 IVREF.
ACMPLP1 IVCMP	No Connection, PORT4 P408, OPAMP1 AMPO Default: No Connection	Select the connection for ACMPLP1 IVCMP.

ACMPLP1 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0, ACMPLP1 IVREF1 Default: No Connection	Select the connection for ACMPLP1 VREF.
OPAMP0 AMPO	No Connection, PORT0 P014, PORT0 P013, PORT0 P003, PORT0 P002 Default: No Connection	Select the connection for OAMP0 AMPO.
OPAMP0 AMPM	No Connection, PORT5 P501, PORT5 500, PORT0 P014, PORT0 P113, PORT0 P003, OPAMP0 AMPO Default: No Connection	Select the connection for OPAMP0 AMPM.
OPAMP0 AMPP	No Connection, PORT5 P500, PORT0 P014, PORT0 P013, PORT0 P002, DAC120 DA Default: No Connection	Select the connection for OPAMP0 AMPP.
OPAMP1 AMPM	No Connection, PORT0 P014, OPAMP1 AMPO Default: No Connection	Select the connection for OPAMP1 AMPM.
OPAMP1 AMPP	No Connection, PORT0 P014, PORT0 P013, PORT0 P003, PORT0 P002, DAC80 DA Default: No Connection	Select the connection for OPAMP1 AMPP.
OPAMP2 AMPM	No Connection, PORT0 P003, OPAMP2 AMPO Default: No Connection	Select the connection for OPAMP2 AMPM.
OPAMP2 AMPP	No Connection, PORT0 P003, PORT0 P002, DAC81 DA Default: No Connection	Select the connection for OPAMP2 AMPP.

Note

The example settings and defaults are for a project using the Synergy S1JA MCU Group. Other MCUs may have different default values and available configuration settings.

Analog Connection HAL Module Clock Configuration

The Analog Connection HAL module needs no clock. Clocks are supplied by the individual analog modules.

Analog Connection HAL Module Pin Configuration

The Analog Connection HAL module has no pins to configure. They are configured by the individual analog modules as needed.

4.2.1.6 Using the Analog Connection HAL Module in an Application

Once the analog connect module has been configured and the files generated, it is ready to be used in an application.

The typical steps in using the analog connect HAL module in an application are:

1. Initialize the analog connect module using analog_connect_api_t::init.
2. If analog connections are selected in the configurator, a connection table with a user configurable name (Connection Table Name) is generated. To make the configured connections with a module named g_analog_connect0 and a connection table named g_analog_connect0_tabl, call: g_analog_connect0.p_api->connectMultiple(&g_analog_connect0_table);
3. Before calling opamp_api_t::start or comparator_api_t::outputEnable, ensure that all required analog connections are set correctly using analog_connect_api_t::connect or analog_connect_api_t::connectMultiple.
4. [Optional] Connections for ACMPHS or OPAMP can be reconfigured at runtime using analog_connect_api_t::connect or analog_connect_api_t::connectMultiple. To reconfigure connections for ACMPLP after comparator_api_t::outputEnable has been called, first call comparator_api_t::close().

These common steps are illustrated in a typical operational flow diagram in the following figure:

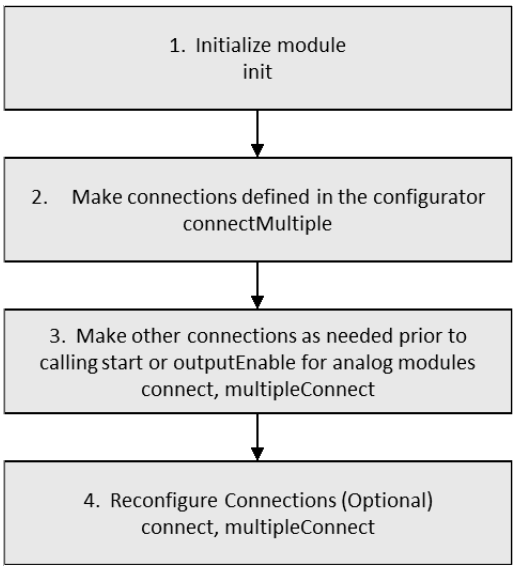


Figure 231: Flow Diagram of a Typical Analog Connect HAL Module Application

4.2.2 Comparator Driver on r_acmphs

4.2.2.1 ACMPHS HAL Module Introduction

The ACMPHS HAL module implements the comparator API for signal processing applications and supports the ACMPHS peripheral available on the Synergy microcontroller hardware. A callback is available to signal the user application on transition events.

ACMPHS HAL Module Features

- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option to include comparator output on VCOUT pin

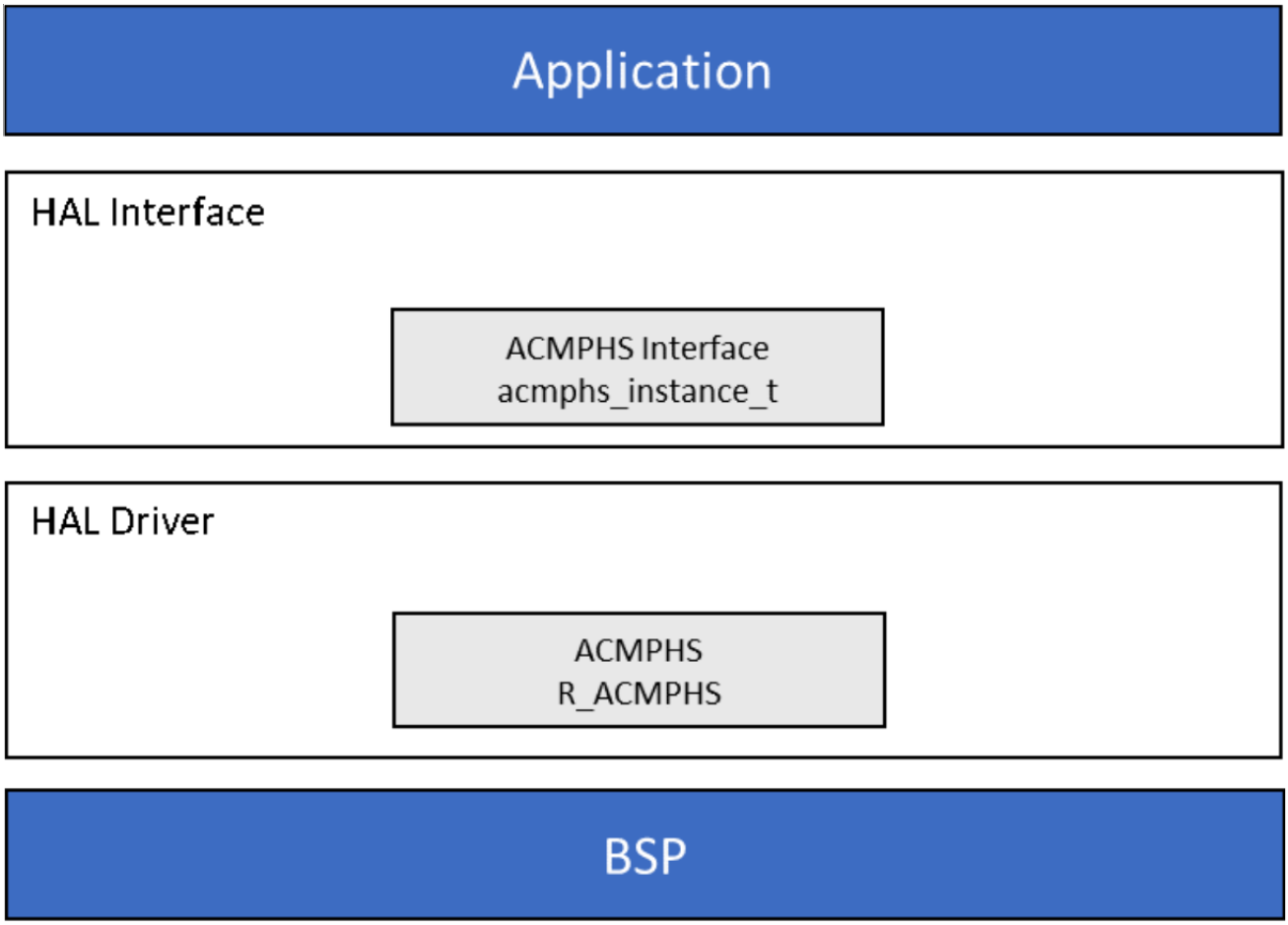


Figure 232: ACMPHS HAL Module Block Diagram

ACMPHS Hardware support details

The following hardware features are, or are not, supported by the SSP for the ACMPHS:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Query HS Comparator result	Callback on rising edge, falling edge, or both	Configurable debounce filter	Option to include comparator output on VCOUT
S124	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A
S1JA	✓	✓	✓	✓
S3A1	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A
S3A7	✓	✓	✓	✓
S5D3	✓	✓	✓	✓
S5D5	✓	✓	✓	✓
S5D9	✓	✓	✓	✓
S7G2	✓	✓	✓	✓

4.2.2.2 ACMPHS HAL Module APIs Overview

The ACMPHS HAL module defines APIs to open, enable, get status and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ACMPHS HAL Module API Summary

Function Name	Example API Call and Description
open	g_comparator0.p_api->open(g_comparator0.p_ctrl, g_comparator0.p_cfg); Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. outputEnable() should be called after the output has stabilized.
outputEnable	g_comparator0.p_api->outputEnable(g_comparator0.p_ctrl); Enables the comparator output, which can be polled using statusGet(). Also enables pin output and interrupts as configured during open().
infoGet	g_comparator0.p_api->infoGet(g_comparator0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds.
statusGet	g_comparator0.p_api->statusGet(g_comparator0.p_ctrl, p_status); Provides the operating status of the comparator.

close	g_comparator0.p_api->close(g_comparator0.p_ctrl); Close the module.
versionGet	g_comparator0.p_api->read(&version); Retrieves the version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.2.3 ACMPHS HAL Module Operational Overview

The ACMPHS HAL module controls the high-speed analog comparator peripheral on a Synergy microcontroller. It directly controls the ACMPHS hardware without using any RTOS elements and provides convenient APIs to simplify development.

ACMPHS HAL Module Important Operational Notes and Limitations

ACMPHS HAL Module Operational Notes

Comparator Output on VCOUT Pin

The signal on the VCOUT pin is a logical 'OR' of the outputs of all comparators (ACMPHS and ACMPLP) with their output pin enabled.

Interrupts and Callbacks

When a comparator event occurs, the Comparator Driver on r_acmphs HAL module calls the callback (comparator_cfg_t::p_callback) with the callback argument (comparator_callback_args_t).

ACMPHS HAL Module Limitations

- This module only works for selected Synergy MCUs. Consult the release notes for your

current SSP release to see which MCUs are supported by this module. The MCU Hardware Manual shows which peripherals are available.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.2.4 Including the ACMPHS HAL Module in an Application

This section describes how to include the ACMPHS HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Comparator Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Comparator Driver is g_acmphs0. This name can be changed in the associated Properties window.)

ACMPHS HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_comparator0 Comparator Driver on r_acmphs	Threads	**New Stack> Driver> Analog> Comparator Driver on r_acmphs**

When the Comparator Driver on r_acmphs is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

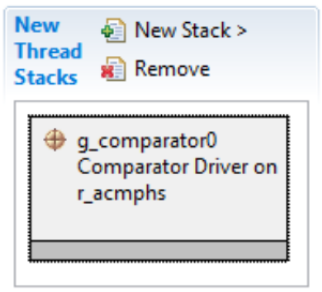


Figure 233: ACMPHS HAL Module Stack

4.2.2.5 Configuring the ACMPHS HAL Module

The ACMPHS HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only

properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ACMPHS HAL Module on r_acmphs

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_comparator	Module name.
Channel	0	Select the hardware channel.
Trigger Edge	Rising, Falling, Both Edge Default: Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Debounce Filter	No Filter, 8, 16, 32 Default: No Filter	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Invert	Not Inverted, Inverted Default: Not Inverted	Turns this on to invest comparator output. This affects the output read from StatusGet(), the pin output level, and the edge trigger.
Pin Output	Disabled, Enabled Default: Disabled	Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is ORed with output from all other ACMPHS and ACMPLP comparators.
Callback	NULL	Define this function in the application. It is called when the Trigger event occurs.

Comparator Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the comparator interrupt.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

ACMPHS HAL Module Clock Configuration

The ACMPHS HAL module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

ACMPHS HAL Module Pin Configuration

To use the ACMPHS HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the ACMPHS HAL Module on r_acmphs

Resource	ISDE Tab	Pin selection Sequence
ACMPHS	Pins	Select **Peripherals**> **Analog:ACMP**

4.2.2.6 Using the ACMPHS HAL Module in an Application

The typical steps in using the ACMPHS HAL module in an application are:

1. Initialize the ACMPHSw using the comparator_api_t::open API.
2. Before enabling the output, consult the hardware manual to configure the internal connections by setting the COMPSELn registers directly. If the internal reference voltage is used, set COMPMDR.CiVRF.
3. After configuring the modules and internal connections, wait for the minimum stabilization wait time before enabling output. The minimum stabilization wait time can be queried using the comparator_api_t::infoGet API.
4. Enable the comparator output using the comparator_api_t::outputEnable API. This enables pin output, interrupts and the comparator_api_t::statusGet API as configured during the comparator_api_t::open API call.
5. [Optional] Use the comparator_api_t::statusGet API to poll comparator status.
6. [Optional] Use the comparator_api_t::close API to disable the comparator and power down the peripheral.

These common steps are illustrated in a typical operational flow diagram in the following figure:

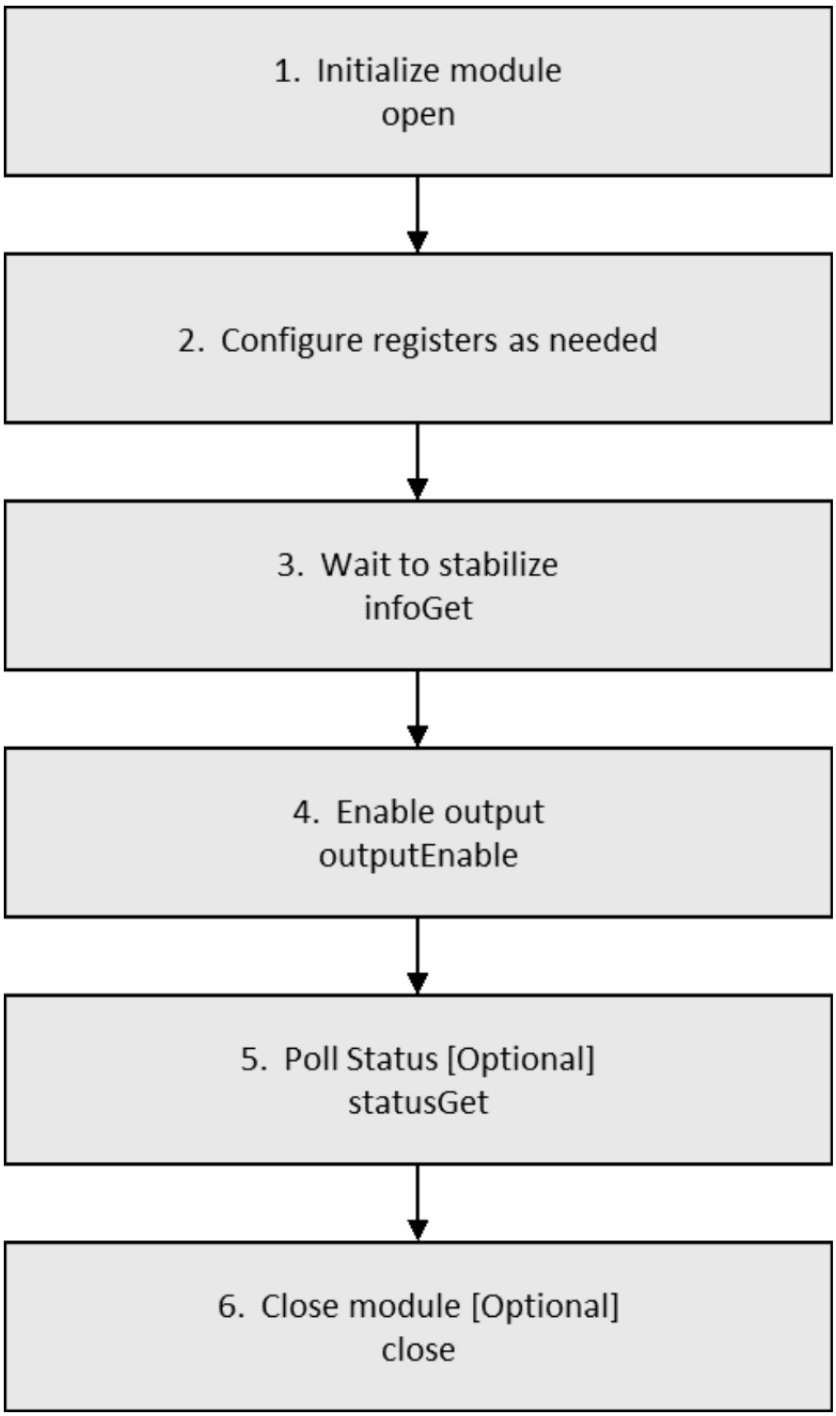


Figure 234: Flow Diagram of a Typical ACMPHS HAL Module Application

4.2.3 Comparator Driver on r_acmplp

4.2.3.1 ACMPLP HAL Module Introduction

The ACMPLP HAL module implements the comparator API for signal processing applications and supports the ACMPLP peripheral available on the Synergy microcontroller hardware. A callback is available to signal the user application on transition events.

ACMPLP HAL Module Features

- Normal mode or window mode
- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option to include comparator output on VCOUT pin

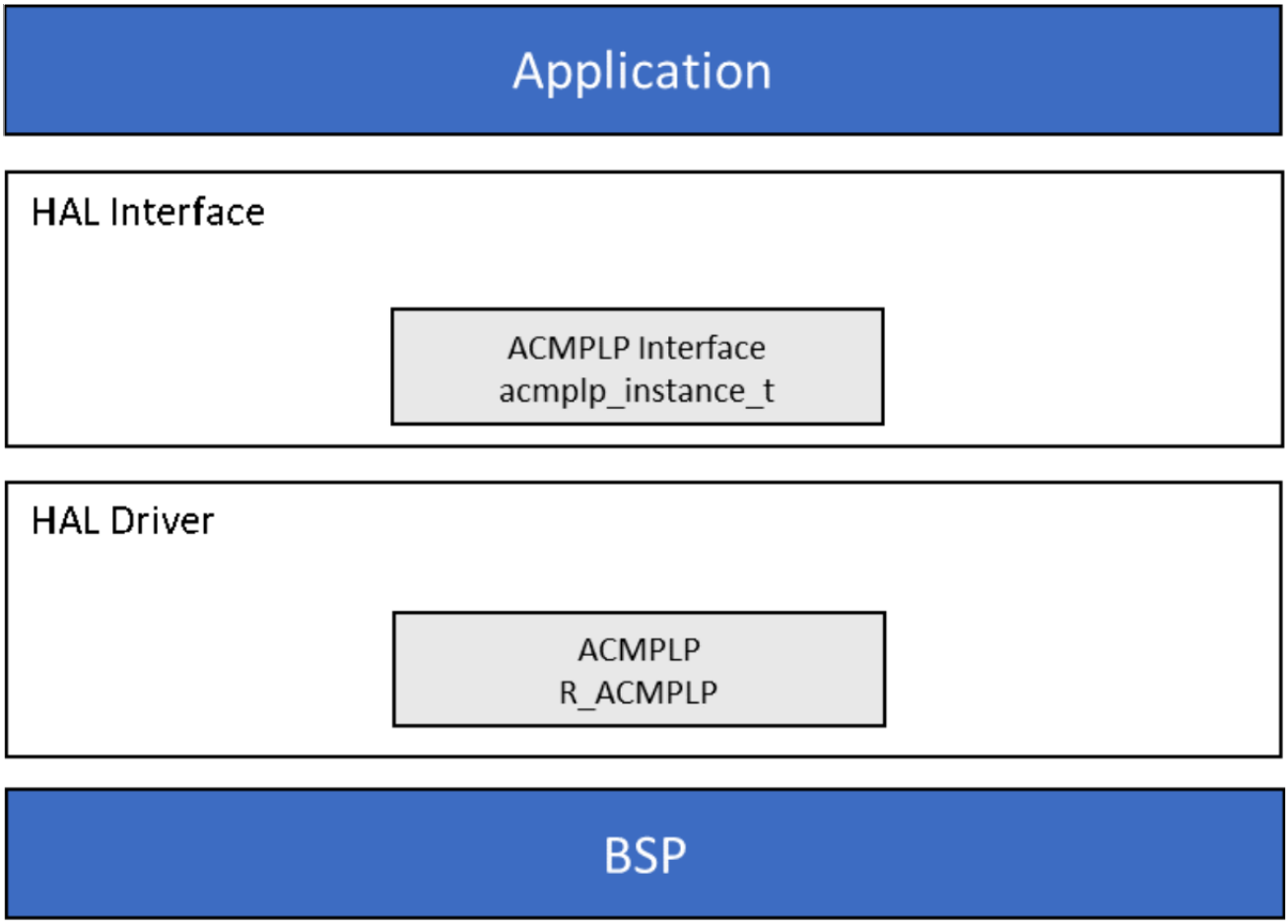


Figure 235: ACMPLP HAL Module Block Diagram

ACMPLP Hardware support details

The following hardware features are, or are not, supported by the SSP for the ACMPLP:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Normal or window mode	Callback on rising edge, falling edge, or both	Configurable debounce filter	Option to include comparator output on VCOUT
S124	✓	✓	✓	✓
S128	✓	✓	✓	✓
S1JA	✓	✓	✓	✓
S3A1	✓	✓	✓	✓
S3A3	✓	✓	✓	✓
S3A6	✓	✓	✓	✓
S3A7	✓	✓	✓	✓
S5D3	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A

4.2.3.2 ACMPLP HAL Module APIs Overview

The ACMPLP HAL module defines API functions to open, enable, get status and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ACMPLP HAL Module API Summary

Function Name	Example API Call and Description
open	g_comparator0.p_api->open(g_comparator0.p_ctrl, g_comparator0.p_cfg); Configures the comparator and starts operation. Callbacks and pin output are not active until comparator_api_t::outputEnable is called. comparator_api_t::outputEnable should be called after the output has stabilized.
outputEnable	g_comparator0.p_api->outputEnable(g_comparator0.p_ctrl); Enables the comparator output, which can be polled using statusGet(). Also enables pin output and interrupts as configured during open().
infoGet	g_comparator0.p_api->infoGet(g_comparator0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds.
statusGet	g_comparator0.p_api->statusGet(g_comparator0.p_ctrl, p_status); Provides the operating status of the comparator.

close	g_comparator0.p_api->close(g_comparator0.p_ctrl); Close the module.
versionGet	g_comparator0.p_api->read(&version); Retrieves the version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.3.3 ACMPLP HAL Module Operational Overview

The Comparator Driver (r_acmplp) HAL module controls the Low-Power Analog Comparator (ACMPLP) peripheral on a Synergy microcontroller. It directly controls the ACMPLP hardware without using any RTOS elements and provides convenient APIs to simplify development.

ACMPLP HAL Module Important Operational Notes and Limitations

ACMPLP HAL Module Operational Notes

Comparator Output on VCOUT Pin

The signal on the VCOUT pin is a logical 'OR' of the outputs of all comparators (ACMPHS and ACMPLP) with their output pin enabled.

Interrupts and Callbacks

When a comparator event occurs, the R_ACMPLP HAL module calls the callback (comparator_cfg_t::p_callback) with the callback argument (comparator_callback_args_t).

ACMPLP HAL Module Limitations

- This module does not support the Filter 16 option for PCLK divisor in the hardware digital

debounce filter.
- This module only works for selected Synergy MCUs. Consult the release notes for your current SSP release to see which MCUs are supported by this module. The MCU Hardware Manual shows which peripherals are available for each MCU.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.3.4 Including the ACMPLP HAL Module in an Application

This section describes how to include the ACMPLP HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Comparator Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Comparator Driver is g_acmplp0. This name can be changed in the associated Properties window.)

ACMPLP HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_comparator0 Comparator Driver on r_acmplp	Threads	**New Stack> Driver> Analog> Comparator Driver on r_acmplp**

When the Comparator Driver on r_acmplp is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

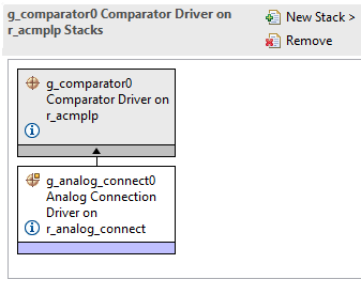


Figure 236: ACMPLP Module Stack

4.2.3.5 Configuring the ACMPLP HAL Module

The ACMPLP HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties

tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ACMPLP HAL Module on r_acmplp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_comparator	Module name.
Channel	0	Select the hardware channel.
Mode	Mode Normal, Mode Window Default: Mode Normal	In normal mode, comparator output is high if VCMP > VREF. In window mode, comparator output is high if VCMP is outside the range of VREF0 to VREF1.
Trigger	Trigger Rising, Trigger Falling, Trigger Both Edge Default: Trigger Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Filter	Filter Off, Filter 1, Filter 8, Filter 32 Default: Filter Off	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Invert	Off, On Default: Off	Turns this on to invest comparator output. This affects the output read from StatusGet(), the pin output level, and the edge trigger.
Pin Output	Off, On Default: Off	Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is OR'd with output from all other ACMPHS and ACMPLP comparators.

Callback	NULL	Define this function in the application. It is called when the Trigger event occurs.
Comparator Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the comparator interrupt.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the ACMPLP HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the Red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Analog Connection HAL Module on r_analog_connect

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_analog_connect0	Module name.
Connection Table Name	g_analog_connect0_table	This connection table name is passed to the connectMultiple API function to make all configured connections.
ACMPHS0 IVCMP	No Connection, PORT5 P500, PORT0 P013, PORT1 P100 Default: No Connection	Select the connection for ACMPHS0 IVCMP *Note* *Not required for ACMPLP Module.*
ACMHS0 IVREF	No Connection, PORT5 P501, PORT0 P014, PORT1 P101, DAC80 DA, DAC120 DA, Analog0 VREF Default: No Connection	Select the connection for ACMPHS0 IVREF *Note* *Not required for ACMPLP Module.*
ACMPL0 IVREF0	No Connection, PORT1 P109, DAC80 DA Default: No Connection	Select the connection for ACMPL0 IVREF0.

ACMPLP1 IVREF1	No Connection, PORT1 P110, DAC81 DA Default: No Connection	Select the connection for ACMPLP1 IVREF1.
ACMPLP0 IVCMP	No Connection, PORT4 P400, OPAMP0 AMPO Default: No Connection	Select the connection for ACMPLP0 IVCMP.
ACMPLP0 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0 Default: No Connection	Select the connection for ACMPLP0 IVREF.
ACMPLP1 IVCMP	No Connection, PORT4 P408, OPAMP1 AMPO Default: No Connection	Select the connection for ACMPLP1 IVCMP.
ACMPLP1 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0, ACMPLP1 IVREF1 Default: No Connection	Select the connection for ACMPLP1 VREF.
OPAMP0 AMPO	No Connection, PORT0P014, PORT0 P013, PORT0P003, PORT0 P002 Default: No Connection	Select the connection for OAMP0 AMPO. *Note* *Not required for ACMPLP Module.*
OPAMP0 AMPM	No Connection, PORT5 P501, PORT5 500, PORT0 P014, PORT0 P113, PORT0 P003, OPAMP0 AMPO Default: No Connection	Select the connection for OPAMP0 AMPM. *Note* *Not required for ACMPLP Module.*
OPAMP0 AMPP	No Connection, PORT5 P500, PORT0 P014, PORT0P013, PORT0 P002, DAC120 DA Default: No Connection	Select the connection for OPAMP0 AMPP. *Note* *Not required for ACMPLP Module.*
OPAMP1 AMPM	No Connection, PORT0 P014, OPAMP1 AMPO Default: No Connection	Select the connection for OPAMP1 AMPM. *Note* *Not required for ACMPLP Module.*

OPAMP1 AMPP	No Connection, PORT0 P014, PORT0 P013, PORT0 P003, PORT0 P002, DAC80 DA Default: No Connection	Select the connection for OPAMP1 AMPP. *Note* *Not required for ACMPLP Module.*
OPAMP2 AMPM	No Connection, PORT0 P003, OPAMP2 AMPO Default: No Connection	Select the connection for OPAMP2 AMPM. *Note* *Not required for ACMPLP Module.*
OPAMP2 AMPP	No Connection, PORT0 P003, PORT0 P002, DAC81DA Default: No Connection	Select the connection for OPAMP2 AMPP. *Note* *Not required for ACMPLP Module.*

Note

The example settings and defaults are for a project using the Synergy S1JA MCU Group. Other MCUs may have different default values and available configuration settings.

ACMPLP HAL Module Clock Configuration

The ACMPLP HAL module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

ACMPLP HAL Module Pin Configuration

To use the ACMPLP HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the ACMPLP HAL Module on r_acmplp

Resource	ISDE Tab	Pin selection Sequence
ACMPLP	Pins	Select **Peripherals**> **Analog:ACMP**

Configuration Settings for the ACMPLP HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the Red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Analog Connection HAL Module on r_analog_connect

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_analog_connect0	Module name.
Connection Table Name	g_analog_connect0_table	This connection table name is passed to the connectMultiple API function to make all configured connections.
ACMPHS0 IVCMP	No Connection, PORT5 P500, PORT0 P013, PORT1 P100 Default: No Connection	Select the connection for ACMPHS0 IVCMP. *Note* *Not required for ACMPLP Module.*
ACMHS0 IVREF	No Connection, PORT5 P501, PORT0 P014, PORT1 P101, DAC80 DA, DAC120 DA, Analog0 VREF Default: No Connection	Select the connection for ACMPHS0 IVREF. *Note* *Not required for ACMPLP Module.*
ACMPL0 IVREF0	No Connection, PORT1 P109, DAC80 DA Default: No Connection	Select the connection for ACMPL0 IVREF0.
ACMPLP1 IVREF1	No Connection, PORT1 P110, DAC81 DA Default: No Connection	Select the connection for ACMPLP1 IVREF1.
ACMPLP0 IVCMP	No Connection, PORT4 P400, OPAMP0 AMPO Default: No Connection	Select the connection for ACMPLP0 IVCMP.
ACMPLP0 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0 Default: No Connection	Select the connection for ACMPLP0 IVREF.
ACMPLP1 IVCMP	No Connection, PORT4 P408, OPAMP1 AMPO Default: No Connection	Select the connection for ACMPLP1 IVCMP.

ACMPLP1 IVREF	No Connection, ANALOG0 VREF, ACMPLP0 IVREF0, ACMPLP1 IVREF1 Default: No Connection	Select the connection for ACMPLP1 VREF.
OPAMP0 AMPO	No Connection, PORT0P014, PORT0 P013, PORT0P003, PORT0 P002 Default: No Connection	Select the connection for OAMP0 AMPO. *Note* *Not required for ACMPLP Module.*
OPAMP0 AMPM	No Connection, PORT5 P501, PORT5 500, PORT0 P014, PORT0 P113, PORT0 P003, OPAMP0 AMPO Default: No Connection	Select the connection for OPAMP0 AMPM. *Note* *Not required for ACMPLP Module.*
OPAMP0 AMPP	No Connection, PORT5 P500, PORT0 P014, PORT0P013, PORT0 P002, DAC120 DA Default: No Connection	Select the connection for OPAMP0 AMPP. *Note* *Not required for ACMPLP Module.*
OPAMP1 AMPM	No Connection, PORT0 P014, OPAMP1 AMPO Default: No Connection	Select the connection for OPAMP1 AMPM. *Note* *Not required for ACMPLP Module.*
OPAMP1 AMPP	No Connection, PORT0 P014, PORT0 P013, PORT0 P003, PORT0 P002, DAC80 DA Default: No Connection	Select the connection for OPAMP1 AMPP. *Note* *Not required for ACMPLP Module.*

OPAMP2 AMPM	No Connection, PORT0 P003, OPAMP2 AMPO Default: No Connection	Select the connection for OPAMP2 AMPM. *Note* *Not required for ACMPLP Module.*
OPAMP2 AMPP	No Connection, PORT0 P003, PORT0 P002, DAC81DA Default: No Connection	Select the connection for OPAMP2 AMPP. *Note* *Not required for ACMPLP Module.*

Note

The example settings and defaults are for a project using the Synergy S1JA MCU Group. Other MCUs may have different default values and available configuration settings.

4.2.3.6 Using the ACMPLP HAL Module in an Application

The typical steps in using the ACMPLP HAL module in an application are:

1. Initialize the ACMPLP using the comparator_api_t::open API.
2. Before VCOUT is enabled, the low level corresponding hardware register must be configured as mentioned below:
    a. Either directly configuring HW registers (COMPMDR/COMPSELn) in application.
    b. Or using Analog Connect module APIs which in turn configure low level registers.
3. After configuring the modules and internal connections, wait for the minimum stabilization wait time before enabling output. The minimum stabilization wait time can be queried using the comparator_api_t::infoGet API.
4. Enable the comparator output using the comparator_api_t::outputEnable API. This enables pin output, interrupts and the comparator_api_t::statusGet API as configured during the open API call.
5. [Optional] Use the comparator_api_t::statusGet API to poll comparator status.
6. [Optional] Use the comparator_api_t::close API to disable the comparator and power down the peripheral.

These common steps are illustrated in a typical operational flow diagram in the following figure:

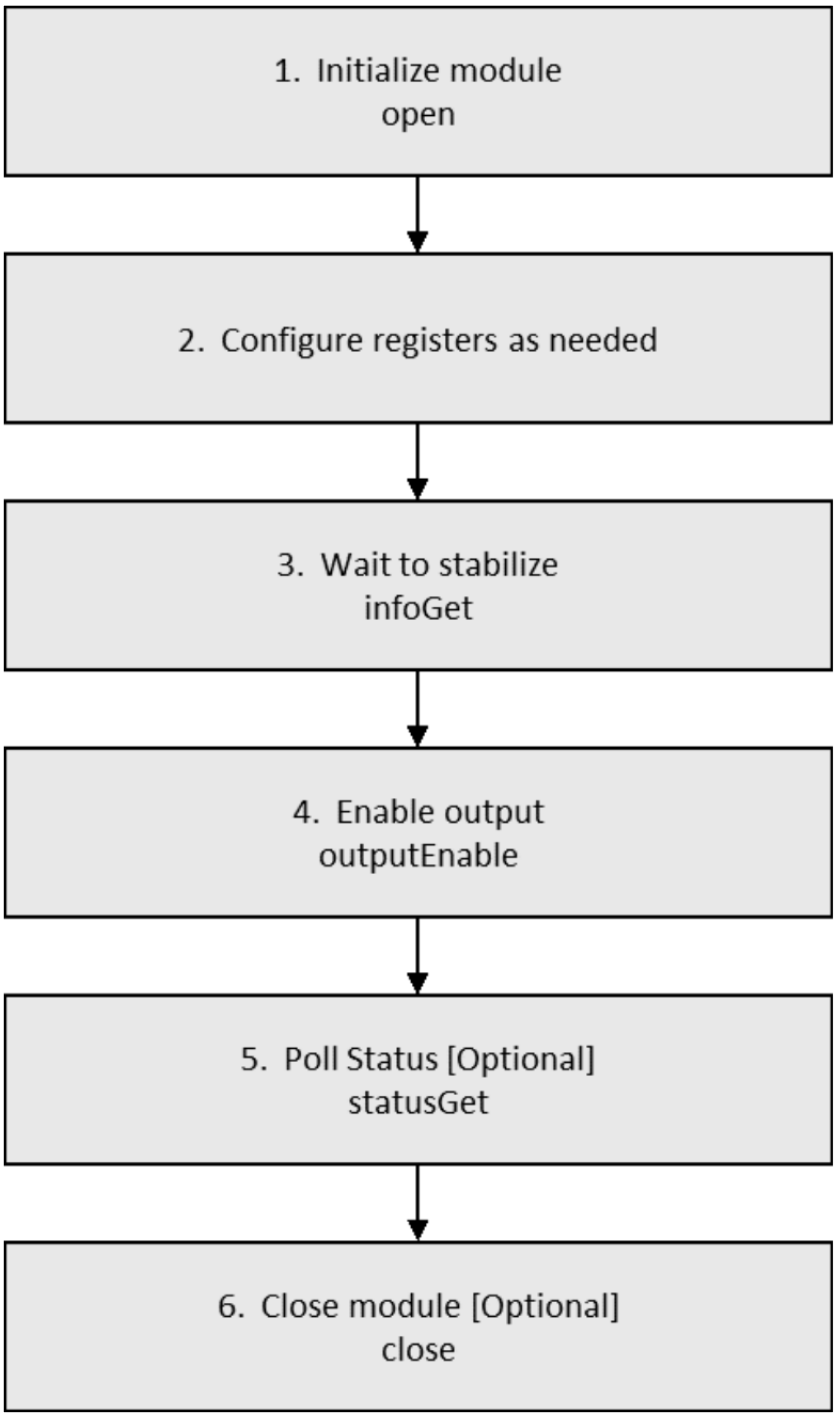


Figure 237: Flow Diagram of a Typical ACMPLP HAL Module Application

4.2.4 ADC Driver

4.2.4.1 ADC HAL Module Introduction

The ADC HAL module implements an API for analog-to-digital conversion applications. It supports the ADC12, ADC14, and ADC16 (for supported MCUs) for the associated peripherals available on Synergy MCUs. A user-defined callback can be used to process the data each time a new sample is complete.

ADC HAL Module Features

- 16-Bit A/D Converter (S1JA)
- 14-Bit A/D Converter (S3A7, S3A3, S3A6, S3A1, S128, S124)
- 12-Bit A/D Converter (S7G2, S5D9, S5D5)
- Multiple Operation Modes
    - Single Scan
    - Group Scan
    - Continuous Scan
- Multiple Channels
    - All analog channels on MCU
        - 13 channels (unit 0) or 12 channels (unit 1) for S7G2
        - 17 channels for S1JA
        - 18 channels for S124
        - 28 channels for S3A7
    - Temperature sensor channel
    - Voltage sensor channel
- Reference voltage selection on 16-Bit A/D Converter (S1JA).
- Programmable gain amplifier (PGA) (S7G2, S5D9, S5D3)
    - Single ended input mode
    - Differential input mode

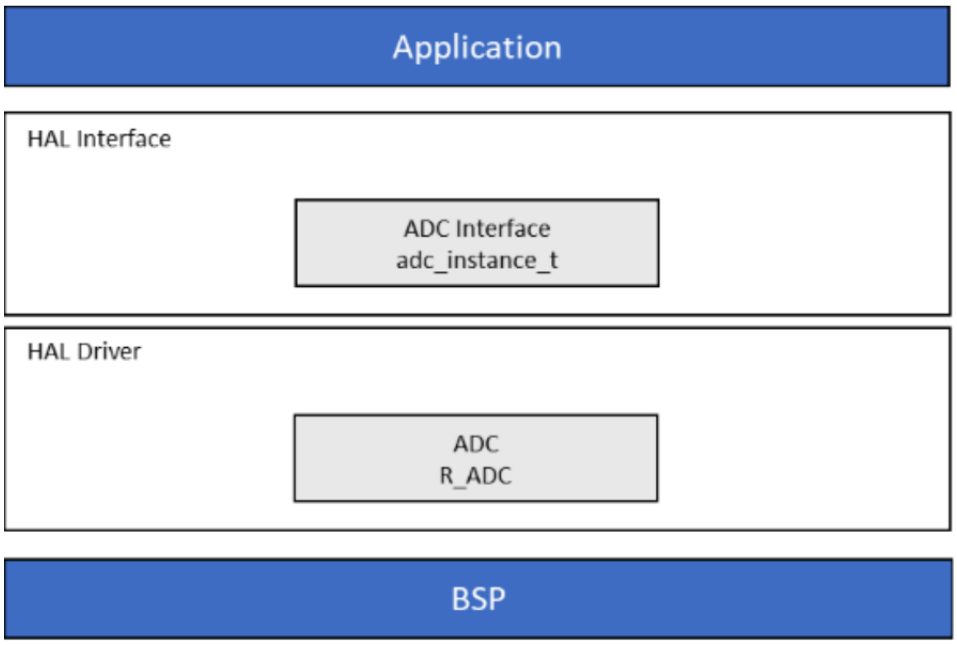


Figure 238: ADC HAL Module Block Diagram

ADC Hardware support details

The following hardware features are, or are not, supported by the SSP for the ADC:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Support for all Analog Channels (Unit 0 for all MCUs and Unit 0 & 1 for S5 and S7 series)	8-Bit	10-bit	12-Bit	14-bit	16-bit	Single scan Mode
S124	✓	N/A	N/A	✓	✓	N/A	✓
S128	✓	N/A	N/A	✓	✓	N/A	✓
S1JA	✓	N/A	N/A	N/A	N/A	✓	✓
S3A1	✓	N/A	N/A	✓	✓	N/A	✓
S3A3	✓	N/A	N/A	✓	✓	N/A	✓
S3A6	✓	N/A	N/A	✓	✓	N/A	✓
S3A7	✓	N/A	N/A	✓	✓	N/A	✓
S5D3	✓	✓	✓	✓	N/A	N/A	✓
S5D5	✓	✓	✓	✓	N/A	N/A	✓
S5D9	✓	✓	✓	✓	N/A	N/A	✓
S7G2	✓	✓	✓	✓	N/A	N/A	✓

MCU Group	Continuous Scan Mode	Group Scan Mode	Programmable Gain Amplifier	Event link function through ELC HAL driver*
S124	✓	✓	N/A	✓
S128	✓	✓	N/A	✓
S1JA	✓	✓	N/A	✓
S3A1	✓	✓	N/A	✓
S3A3	✓	✓	N/A	✓
S3A6	✓	✓	N/A	✓
S3A7	✓	✓	N/A	✓
S5D3	✓	✓	✓	✓
S5D5	✓	✓	N/A	✓
S5D9	✓	✓	✓	✓

S7G2	✓	✓	✓	✓

- Note: ELC is supported but only for Group mode. This must be set up manually by programming the Event Link Controller.

4.2.4.2 ADC HAL Module APIs Overview

The ADC HAL module defines APIs to open, configure scans, start scans, stop scans, read the conversion results of the ADC scans, and close the ADC unit. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ADC HAL Module API Summary

Function Name	Example API Call and Description
open	g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg); Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.
scanCfg	g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.p_channel_cfg); Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call.
scanStart	g_adc.p_api->scanStart(g_adc.p_ctrl); Start the scan (in case of a software trigger), or enable the hardware trigger.
scanStop	g_adc.p_api->scanStop(g_adc.p_ctrl); Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.
scanStatusGet	g_adc.p_api->scanStatusGet(g_adc.p_ctrl); Check scan status.
read	g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result(s).
sampleStateCountSet	g_adc.p_api->sampleStateCountSet(g_adc.p_ctrl ,&adc_sample); Set the sample state count for the specified channel.
close	g_adc.p_api->close(g_adc.p_ctrl); Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

infoGet	g_adc.p_api->infoGet(g_adc.p_ctrl, &adc_info); Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read for the DTC/DMAC to read the conversion results of all configured channels.
versionGet	g_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform R_ADC_Close first.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.4.3 ADC HAL Module Operational Overview

The ADC driver on r_adc HAL module controls the ADC peripherals on a Synergy microcontroller, as configured by the user. It directly controls the ADC hardware without using any RTOS elements. It provides convenient API functions to simplify development.

The driver supports three operation modes: single-scan, continuous-scan, and group-scan modes.

Single-scan Mode

In single scan mode, one or more specified channels are scanned once per trigger. A channel bit-mask is used in the channel properties configuration settings to indicate the scanned channels. Single-scan mode sequentially converts the analog inputs of the selected channels in the ascending order of the channel number. A callback event is generated after all selected channels have completed the conversion operation.

Continuous-scan Mode

Continuous-scan mode sequentially converts the analog inputs of selected channels continuously in the ascending order of the channel numbers. A single trigger is required to start the scan. No callback is used in this mode and interrupts must be disabled. The scanStatusGet API function is

used to determine when data is available.

Group-scan Mode

Group-scan mode allows the application to allocate channels to one of two groups (A and B). Analog inputs of the selected channels are converted for each group in the ascending order of the channel numbers. Conversion begins when the specified start trigger for that group is received. A callback interrupt is generated after all selected channels in the associated group have completed the conversion operation. The interrupt event indicates which group has completed conversion.

In group mode, only hardware triggers can be used, as opposed to normal mode, where software triggers or an external trigger can be used. With the priority configuration parameter, you can specify:

- Whether a trigger for one group can interrupt an ongoing scan for the other group.
- Whether an interrupted scan resumes or restarts or simply aborts the current scan and waits for the next trigger.

Interrupt and Callback Overview

When a scan or calibration (on supported MCUs) is complete and a callback is provided in the application code, (and if interrupts are enabled) the module invokes the defined callback and provides an argument that indicates the ADC unit, the event, the address of the converted data, and the channel.

The module supports two interrupts:

- The Normal/Group A Interrupt (Scan End Interrupt) fires when a scan is completed in single scan mode, when a Group A scan is completed in group mode, or at the end of calibration for supported MCUs.
- The Group B Interrupt (Scan End Group B Interrupt) fires when a group B scan is completed in group mode.

Interrupts function differently in each mode:

- In single-scan mode, the Normal interrupt (Scan End Interrupt) is triggered when the scan is completed.
- In continuous scan-mode, the hardware will constantly scan the selected channels. In this mode, the driver will return an error if interrupts are enabled, so they must be disabled in this mode.
- In group mode, the ADC unit provides two interrupts (when enabled). The Normal interrupt (called Group A interrupt in this mode, even though it is the same vector as the Normal interrupt) and the Group B interrupt. The Group A interrupt (Scan End Interrupt) is triggered when a Group A scan is completed. The Group B interrupt (Scan End Group B Interrupt) is triggered when a Group B scan is completed.

Note

You must change the Scan End Interrupt Priority and Scan End Group B Interrupt Priority configuration setting in the selected units' properties window from the default Disabled setting to the desired Priority level to Enable the associated interrupts.

When Interrupts Are Not Enabled

If interrupts are not enabled, the scanStatusGet API is used to poll the ADC to determine when the scan has completed. The read API function is used to access the converted ADC result.

For MCUs that support calibration, if interrupts are not enabled, the application program must wait 24 ms and then check the status of the calibration function using the infoGet API. Once calibration is complete, another API can be used.

ADC HAL Module Important Operational Notes and Limitations

ADC HAL Module Operational Notes

Sample-State Count Setting

The application program can modify the setting of the sample-state count by calling the adc_api_t::sampleStateCountSet API function. The application program only needs to modify the sample-state count settings from their default values to increase the sampling time. This can be either because the impedance of the input signal is too high to secure sufficient sampling time under the default setting or if the ADCLK is too slow. To modify the sample-state count for a given channel, set the channel number and the number of states when calling the adc_api_t::sampleStateCountSet API function. Valid sample state counts are 7-255.

Note

Although the hardware supports a minimum number of sample states of 5, some Synergy MCUs require 7 states, so the minimum is set to 7. At the lowest supported ADC conversion clock rate (1 MHz), these extra states will lead to, at worst case, a 2 microsecond increase in conversion time. At 60 MHz the extra states will add 33.4 ns to the conversion time.

If the sample state count needs to be changed for multiple channels, the application program must call the adc_api_t::sampleStateCountSet API function repeatedly, with appropriately modified arguments for each channel.

Triggering a Data Transfer with the ADC

To trigger a transfer of data when the ADC scan completes, configure the data transfer with the activation_source set to ELC_EVENT_ADCn_SCAN_END or ELC_EVENT_ADCn_SCAN_END_B (where n is the ADC channel number). The adc_api_t::infoGet API function can be called to retrieve the ADC unit-specific information to use with the transfer API. Refer to the ELC Module Overview for additional information.

Triggering ELC Events with the ADC

The ADC unit can trigger the start of other peripherals by using the ELC. Refer to the ELC Module Overview for additional information.

Using the Temperature Sensor with the ADC

The ADC HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula, T = (Vs – V1)/slope + T1, where:

- T: Measured temperature (°C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (°C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- T2: Temperature at the experimental measurement of another point (°C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/°C); slope = (V2 – V1)/ (T2 – T1)

Note

The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter- TSN Characteristics Table, Temperature slope entry. The slope is positive for S7 and S5 devices and negative for S3 and S1 devices.

ADC HAL Module Limitations

When configuring the module, the temperature and voltage sensors must not be selected if any of the other available channels are also selected. The temperature sensor and the voltage sensor can both be used together, but neither can be used if any of the regular ADC channels are used.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.4.4 Including the ADC HAL Module in an Application

This section describes how to include the ADC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ADC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the ADC Driver is g_adc0. This name can be changed in the associated Properties window.)

ADC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_adc0 ADC Driver on r_adc	Threads	**New Stack> Driver> Analog> ADC Driver on r_adc**

When the ADC Driver on r_adc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

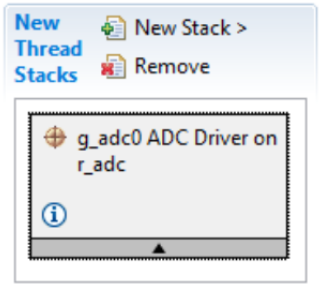


Figure 239: ADC HAL Module Stack

4.2.4.5 Configuring the ADC HAL Module

The ADC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected, code for parameter checking is included in the build.
Name	g_adc0	Module name.
Unit	0, 1 (S7G2, S5D9 and S5D5) Default: 0	Specify the ADC Unit to be used. The S7G2 has two units; 0 and 1.
Resolution (resolution varies by MCU)	14-Bit, 12-Bit, 10-Bit, 8-Bit Default: 8-Bit	Specify the conversion resolution for this unit.
Alignment	Right, Left Default: Right	Specify the conversion result alignment.
Clear after read	Off, On Default: On	Specify if the result register must be automatically cleared after the conversion result is read. *Note* *If this is enabled, then watching the result register using a debugger always results in a 0.*
Mode	Single Scan, Continuous Scan, Group Scan Default: Single Scan	Specify the mode that this ADC unit is used in.
Internal Calibration During Open()	Disabled, Enabled Default: Enabled	Internal calibration during open selection.

Channel 0	- Unused, PGA 0: Configure gain from below field. - Use in Normal/Group A, - Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 1	- Unused, - Use in Normal/Group A, - Use in Group B PGA 1: Configure gain from below field. Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 2	- Unused, - Use in Normal/Group A, - Use in Group B PGA 2: Configure gain from below field. Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled.
Channels 3-13	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 14 (S3 Series Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

Channel 15 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 16-20	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 21 (Unit 0 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 22-24	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 25 (S3 series only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

Channel 26 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 27 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Temperature Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Temperature sensor use selection for Channel Scan Mask.
Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Voltage sensor use selection for Channel Scan Mask.
Normal/Group A Trigger	None, Asynchronous External Trigger 0, ELC Event, Software Default: Software	Specify the trigger type to be used for this unit. If group mode is used adc_cfg_t::mode, then this field is used to set the Group A trigger. *Note* *The only valid option in group mode is the ELC trigger.*
Group B Trigger (Valid Only in Group Scan Mode)	ELC Event (The only valid trigger for either group in Group Scan Mode)	Specify the group B trigger. This option is only valid if group mode is chosen in adc_cfg_t::mode.

Group Priority (Valid only in Group Scan Mode)	Group A cannot interrupt Group B, Group A can interrupt Group B; Group B scan restarts at next trigger, Group A can interrupt Group B; Group B scan restarts immediately, Group A can interrupt Group B; Group B scan restarts immediately and scans continuously Default: Group A cannot interrupt Group B	Determines whether an ongoing group B scan can be interrupted by a group A trigger, whether it should abort on a group A trigger, or if it should pause to allow group A scan and restart immediately after group A scan is complete. *Note* *This field is valid only in group mode.*
Add/Average Count	Disabled, Add two samples, Add three samples, Add four samples, Add sixteen samples, Average two samples, Average four samples, Average eight, Average sixteen Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask adc_channel_cfg_t::add_mask. Average eight and Average sixteen options are applicable only for S1JA. Add count option is not applicable for S1JA.
Channels 0-27	Disabled, Enabled Default: Disabled	This field is valid only if adc_cfg_t::add_average_count is enabled. This field determines what channels results are to be averaged or summed.
Temperature Sensor	Disabled, Enabled Default: Disabled	Temperature sensor use selection for Addition/Averaging Mask.
Voltage Sensor	Disabled, Enabled Default: Disabled	Voltage sensor use selection for Addition/Averaging Mask.
Sample and Hold Mask	Select channels for which individual sample and hold circuit is to be enabled	Sample and hold mask selection.
Channels 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in adc_channel_cfg_t::sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.

Sample Hold States (Applies only to the 3 channels selected above)	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if adc_channel_cfg_t::sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. *Note* *Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.*
Callback	NULL	A user callback function can be registered in adc_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Scan End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Interrupt Priority selection.
Scan End Group B Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Scan End Group B Interrupt Priority selection.
Voltage reference (Only for S1JA)	- External VREFH0 - Internal VREF 1.5V - Internal VREF 2.0V - Internal VREF 2.5V	The ADC module will use the selected one as reference voltage.
Over-current protection (Only for S1JA)	- Enable - Disabled	Enables or disable over-current detection on sensor module.
Programmable Gain Amplifier	Select PGA channel from 'Channel Mask Scan' and Gain from below options	Select PGA channel from 'Channel Mask Scan' and Gain from below options.

PGA 0	- Disabled - Single Input_x2 - Single Input_x2.5 - Single Input_x_2.66 - Single Input_x_2.85 - Single Input_x_3.07 - Single Input_x_3.33 - Single Input_x_3.63 - Single Input_x_4.00 - Single Input_x_4.44 - Single Input_x_5.00 - Single Input_x_5.71 - Single Input_x_6.66 - Single Input_x_8.00 - Single Input_x_10.0 - Single Input_x_13.33 - Diff Input_x_1.5" - Diff Input_x_2.3" - Diff Input_x_4.0" - Diff Input_x_5.66"	Applicable only for S7G2, S5D9, S5D3.
PGA 1	- Disabled - Single Input_x2 - Single Input_x2.5 - Single Input_x_2.66 - Single Input_x_2.85 - Single Input_x_3.07 - Single Input_x_3.33 - Single Input_x_3.63 - Single Input_x_4.00 - Single Input_x_4.44 - Single Input_x_5.00 - Single Input_x_5.71 - Single Input_x_6.66 - Single Input_x_8.00 - Single Input_x_10.0 - Single Input_x_13.33 - Diff Input_x_1.5" - Diff Input_x_2.3" - Diff Input_x_4.0" - Diff Input_x_5.66"	Applicable only for S7G2, S5D9, S5D3.

PGA 2	- Disabled - Single Input_x2 - Single Input_x2.5 - Single Input_x_2.66 - Single Input_x_2.85 - Single Input_x_3.07 - Single Input_x_3.33 - Single Input_x_3.63 - Single Input_x_4.00 - Single Input_x_4.44 - Single Input_x_5.00 - Single Input_x_5.71 - Single Input_x_6.66 - Single Input_x_8.00 - Single Input_x_10.0 - Single Input_x_13.33 - Diff Input_x_1.5" - Diff Input_x_2.3" - Diff Input_x_4.0" - Diff Input_x_5.66"	Applicable only for S7G2, S5D9, S5D3.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

ADC HAL Module Clock Configuration

The ADC HAL module uses the PCLKC as its clock source (ADCLK.) The only restriction when configuring this clock is that it should be set to less than the max ADC clock; there is also a restriction on the ratio of the PCLKC and PCLKB clocks specified in the hardware manual.

The ADC-conversion time depends on the PCLKC setting.

To set the PCLKB and PCLKC frequencies, use the clock configurator in the ISDE.

To change the clock frequency at run-time, use the CGC Interface.

ADC HAL Module Pin Configuration

To use the ADC HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the ADC HAL Module on r_adc

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select **Peripherals**> **Analog Pins**> **ADC0\1**> **AN_XX**

4.2.4.6 Using the ADC HAL Module in an Application

The typical steps in using the ADC HAL module in an application are:

1. Initialize the ADC using the adc_api_t::open API. (If calibration is enabled in the configuration, it will be performed as a part of the open call for the MCUs that support calibration.)
2. Configure the channels using the adc_api_t::scanCfg API. For MCUs that support calibration, if the calibration was not enabled in the configuration, then it must be performed before starting the first scan. Start the calibration (for supported MCUs) using the adc_api_t::calibrate API.
    a. If interrupts are disabled wait for at least 24 ms (for 32 MHz PCLKB), check status using the adc_api_t::infoGet API to insure the calibration is complete before using other ADC APIs.
    b. If interrupts are enabled, the callback will be invoked when the calibration is complete.
3. Start a conversion using the desired trigger with the adc_api_t::scanStart API.
    a. If a hardware trigger is used, this call enables the ADC unit to be triggered by the hardware trigger. If a software trigger is used, then this call starts the ADC scan.
4. If interrupts are disabled, use the adc_api_t::scanStatusGet API to determine if the scan is complete.
5. If interrupts are enabled, the callback will be invoked when the scan is complete.
6. Read the results of the conversion using the adc_api_t::read API.
7. Stop the ADC scan by calling the adc_api_t::scanStop API
    a. This prevents the ADC from being triggered by an external trigger or a hardware trigger; it also forces a stop of a software-triggered scan if one is ongoing.
8. Operate on the received data as needed by the application.
9. Use the adc_api_t::close API to power down the peripheral.

These common steps are illustrated in a typical operational flow diagram in the following figure:

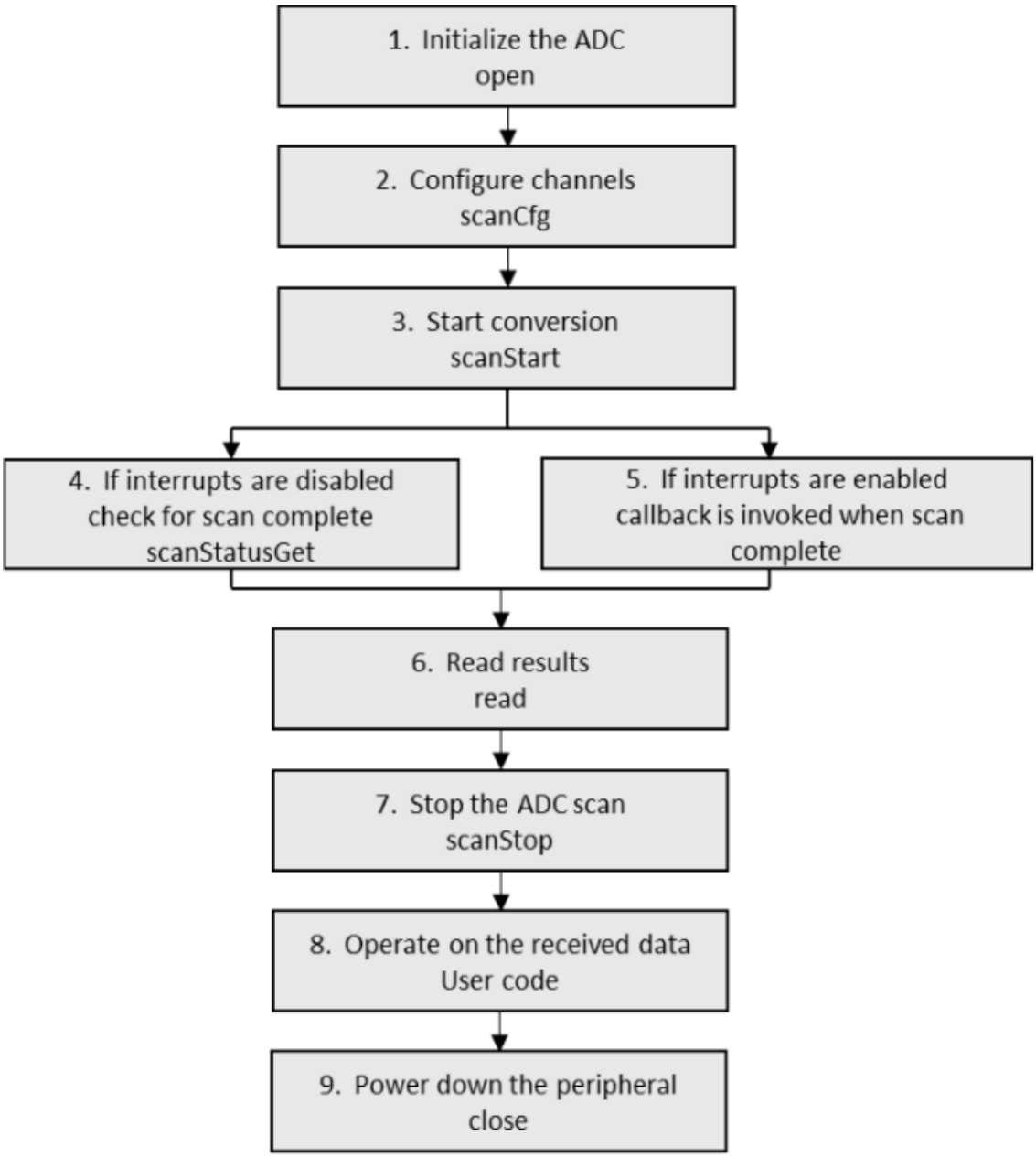


Figure 240: Flow Diagram of a Typical ADC HAL Module Application

4.2.5 Timer Driver on r_agt

4.2.5.1 AGT HAL Module Introduction

The AGT HAL module implements a high-level API for timing applications and uses the AGT peripheral on a Synergy MCU. A user-defined callback can be created to respond to a timer event.

AGT HAL Module Features

- Configures a timer for a set period and generates one of the following events when the period expires:
  - Interrupt the CPU, which calls a user-callback function (if provided)
  - Toggle a port pin
  - Transfer data using DMAC/DTC (if configured with transfer interface)
  - Start another peripheral (if configured with events and peripheral definitions)
- Multiple Channels: 16-bit x 2 channels
  - Channel 1 can be clocked by the channel 0 underflow, creating a cascaded 32-bit timer
- Core Clock: Can be clocked using PCLKB, LOCO, or Fsub. When clocked by LOCO or Fsub, it can be used to wake up the MCU from sleep modes

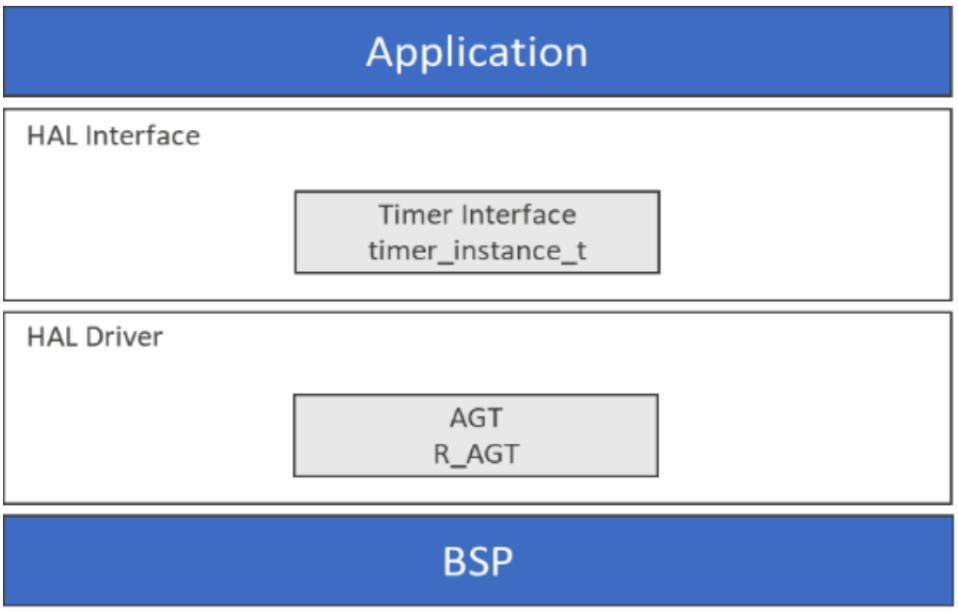


Figure 241: AGT HAL Module Block Diagram

AGT Hardware support details

The following hardware features are, or are not, supported by the SSP for the AGT.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Timer Mode	Pulse Output Mode	Event Counter Mode	Pulse width measurement mode
S124	✓	✓	☒	☒
S128	✓	✓	☒	☒
S1JA	✓	✓	☒	☒

S3A1	✓	✓	☒	☒
S3A3	✓	✓	☒	☒
S3A6	✓	✓	☒	☒
S3A7	✓	✓	☒	☒
S5D3	✓	✓	☒	☒
S5D5	✓	✓	☒	☒
S5D9	✓	✓	☒	☒
S7G2	✓	✓	☒	☒

MCU Group	Pulse period measurement mode	Event link function through ELC HAL driver	Compare/Match Function
S124	☒	☒	✓
S128	☒	☒	✓
S1JA	☒	☒	✓
S3A1	☒	☒	✓
S3A3	☒	☒	✓
S3A6	☒	☒	✓
S3A7	☒	☒	✓
S5D3	☒	☒	✓
S5D5	☒	☒	✓
S5D9	☒	☒	✓
S7G2	☒	☒	✓

4.2.5.2 AGT HAL Module APIs Overview

The AGT HAL module defines APIs for opening, closing, starting and stopping timers. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

AGT HAL Module API Summary

Function Name	Example API Call and Description
open	g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg) Initial configuration.
stop	g_timer0.p_api->stop(g_timer0.p_ctrl) Stop the counter.
start	g_timer0.p_api->start(g_timer0.p_ctrl) Start the counter.

reset	g_timer0.p_api->reset(g_timer0.p_ctrl) Reset the counter initial value.
counterGet	g_timer0.p_api->counterGet(&value) Get current counter value and store it in the provided pointer, value.
periodSet	g_timer0.p_api->periodSet(g_timer0.p_ctrl, period, unit) Set the time until the timer expires.
dutyCycleSet	g_timer0.p_api->dutyCycleSet(g_timer0.p_ctrl, period, unit, pin) Sets the time until the duty cycle expires.
infoGet	g_timer0.p_api->infoGet(&info) Get the time until the timer expires in clock counts and store it in provided pointer, info.
close	g_timer0.p_api->close(g_timer0.p_ctrl) Allows driver to be reconfigured and may reduce power consumption.
versionGet	g_timer0.p_api->versionGet(g_timer0.p_ctrl, &version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Operation is successful.
SSP_ERR_ASSERTION	Parameter is NULL or configuration setting is not allowed.
SSP_ERR_IN_USE	The channel specified is already open.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt is not enabled in the BSP.
SSP_ERROR_NOT_OPEN	The channel is not open.
SSP_ERR_INVALID_ARG	Invalid argument provided.
SSP_ERR_INVALID_HW_CONDITION	Invalid hardware setting detected.
SSP_ERR_INVALID_PTR	A pointer parameter was NULL, but needed a non-NULL value.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.5.3 AGT HAL Module Operational Overview

The AGT HAL module configures a timer to a user-specified period. When the period elapses, the CPU can be interrupted, a port pin can be toggled, a transfer of data using the DMAC or DTC can be initiated, or another peripheral can be triggered to begin operation.

The following figure shows a flowchart for toggling a port pin or generating a CPU interrupt after a specified period. This flowchart is appropriate for both AGT and GPT counters. (Replace the GPT references with AGT references for AGT operation. AGT is a down counter so change overflow to underflow.)

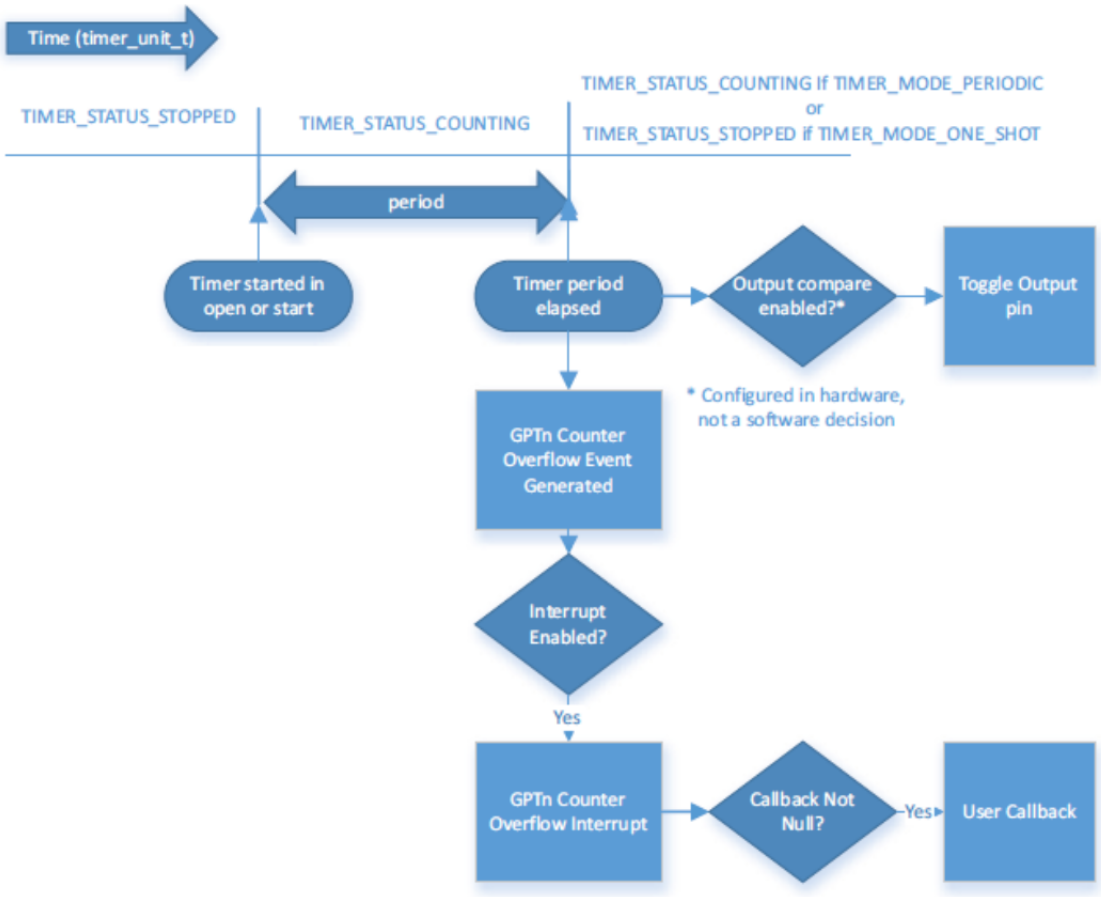


Figure 242: AGT HAL Module Flow Chart

Two different timer modules, the GPT and the AGT, are supported in the SSP. The following sections provide information on both modules so that the developer can compare and contrast the capabilities of each module for a particular application. For additional information on the GPT, refer to the GPT User's Guide.

The GPT module is recommended for most generic timer applications, but either module can be used for a basic timer functionality. The use cases in which one timer module would be preferred over the other are described as follows:

Selecting the GPT Timer Module

The GPT module uses a high-resolution 32-bit counter that can only be clocked by PCLKA. There are more GPT channels than AGT channels on Synergy devices, so using the GPT is less likely to cause a resource conflict.

Selecting the AGT Timer Module

The AGT module uses a 16-bit counter that can be clocked by PCLKB, LOCO or Fsub. If clocked by LOCO or Fsub, the AGT interrupt can be used to wake the MCU from sleep modes. There are two channels, and channel 1 can be clocked by channel 0 underflow, effectively creating a 32-bit cascaded timer.

AGT HAL Module Important Operational Notes and Limitations

AGT HAL Module Operational Notes

The maximum time period depends on the timer type and the input clock frequency.

- On a GPT with 32-bit resolution with PCLKA running at 120 MHz, the maximum period is approximately 36650 seconds, which is just over 10 hours.
- On a GPT with 16-bit resolution with PCLKA running at 32 MHz, the maximum period is approximately 2.09 seconds.
- On an AGT with 16-bit resolution with PCLKB running at 60 MHz, the maximum period is approximately 8.7 ms.
- On an AGT with 16-bit resolution with Fsub or LOCO at 32 kHz as its count source, the maximum period is approximately 262 seconds using pre-scalar up to 128.

The AGT counter underflow interrupt for the selected channel used must be enabled in the BSP in the following situations:

- To get a software interrupt when the timer period has elapsed.
- To use one-shot mode.

When the count source selected as AGTO Fsub or AGTO LOCO, turn on the count source using call cgc_api_t::clockStart API before calling timer_api_t::open.

When the AGTn AGTI interrupt is enabled in the BSP, the corresponding ISR is defined in the timer driver. The ISR calls a user-callback function if one was registered in open.

Note

Interrupts may be skipped when used with the DTC peripheral with the IRQ set to TRANSFER_IRQ_END.

AGT Output Timer Signal

If the timer output is configured, (AGTO Output Enabled set to true) the output pin starts at a high level if the output inverted is configured to True and a low level if it is configured to False. The output pin toggles every time the period elapses, beginning with the first time the period elapses after the timer is started.

In one-shot mode, the output is also configured to toggle when the timer starts counting. This generates a pulse - the timer toggles from the stop level when counting begins and toggles back to the stop level when counting ends.

Timer Period Calculation

The timer period is defined as the time until the timer expires. When output compare is used, the output pin toggles once per period, so the traditional period (from rising edge to rising edge) is twice the period specified in the software.

Runtime period calculation based on the current clock settings is available from timer_api_t::open

and timer_api_t::periodSet.

If the specified timer period is different than the raw counts, the period is calculated using the current timer clock frequency (see Configuring the GPT Clocks or Configuring the AGT Clocks). The current timer clock frequency is determined using systemClockFreqGet. This frequency is used in the appropriate formula from the following table as clk_freq_hz.

Timer Period Calculation

Timer Units	Description
TIMER_UNIT_PERIOD_NSEC	Counts = (period * clk_freq_hz)/ (1000000000 * channel_0_period)
TIMER_UNIT_PERIOD_USEC	Counts = (period * clk_freq_hz)/ (1000000 * channel_0_period)
TIMER_UNIT_PERIOD_MSEC	Counts = (period * clk_freq_hz)/ (1000 * channel_0_period)
TIMER_UNIT_PERIOD_SEC	Counts = (period * clk_freq_hz) / (channel_0_period)
TIMER_UNIT_FREQUENCY_HZ	Counts = (clk_freq_hz)/ (period * channel_0_period)
TIMER_UNIT_FREQUENCY_KHZ	Counts = (clk_freq_hz)/ (1000 * period * channel_0_period)

Note

In normal mode channel_0_period value will be 1 and in cascade mode channel_0_period value will be the timer T0 count value.

Timer Period Calculation

If the requested period is larger than the counter size (32-bit or 16-bit), the driver selects the smallest divisor that allows the result to fit in the counter size. If the counter value is larger than the counter size with the largest divisor (1024), an error code (SSP_ERR_INVALID_ARGUMENT) is returned.

Triggering DMAC/DTC with GPT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_GPTn_COUNTER_OVERFLOW (where n is the GPT channel number). See the DMAC or DTC guides for further information.

Note

If you use the timer in one-shot mode with the DTC, the entire transfer completes before the interrupt stops the timer if the IRQ is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set the IRQ to TRANSFER_IRQ_EACH or use the DMAC for the transfer.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals.

Triggering DMAC/DTC with AGT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_AGTn_AGTI (where n is the AGT channel number). See the Transfer Interface for further information.

Note

If you use the timer in one-shot mode with the DTC, the entire transfer completes before the interrupt stops the timer if irq is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set irq to TRANSFER_IRQ_EACH, or use the DMAC for the transfer.

Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals listed in elc_peripheral_t. (See events and peripheral definitions for further information.)

Cascading AGT Timers to Create a 32-bit timer

AGT Channel 1 can be clocked by the AGT Channel 0 underflow, creating a cascaded 32-bit timer. In this mode, the AGT Channel 0 output frequency will be the input frequency of AGT Channel 1. With cascaded operation, longer time periods are achievable:

- On a cascaded AGT with PCLKB running at 60 MHz, the maximum possible period is approximately 574.1 seconds.
- On a cascaded AGT with LOCO running at 32 kHz, the maximum possible period is approximately 1048560 seconds.

Cascaded Timer Period Calculation

AGT Channel 0 period calculation will be the same as normal timer period calculation as shown in the preceding table. If the requested period is larger than the counter size (32-bit or 16-bit), the driver selects the smallest divisor that allows the result to fit in the counter size.

AGT Channel 1 period calculation will be same as shown in the preceding table, except the "channel_0_period" value will be equal to AGT Channel 0's output period value (that is, after divisor selection).

Note

The cascaded timer output will have the granularity error of 1 / (clock source).

AGT HAL Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.5.4 Including the AGT HAL Module in an Application

This section describes how to include the AGT HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Timer Driver to an application, simply add it to a thread using the stacks selection

sequence given in the following table. (The default name for the Timer Driver is g_agt0. This name can be changed in the associated Properties window.)

AGT HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_agt0 Timer Driver on r_agt	Threads	**New Stack> Driver> Timers> Timer Driver on r_agt**

When the Timer Driver on r_agt is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

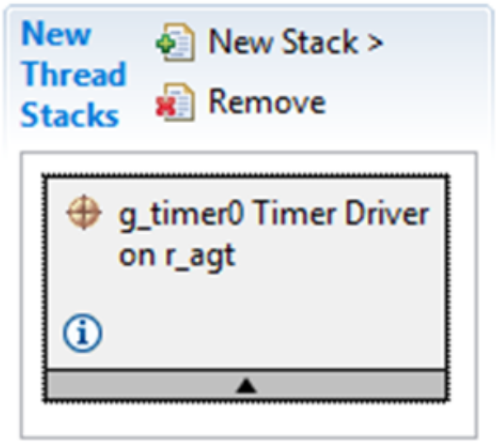


Figure 243: AGT HAL Module Stack

4.2.5.5 Configuring the AGT HAL Module

The AGT HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic, One Shot Default: Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Microseconds	See Timer Period Calculation.
Auto Start	True, False Default: True	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.

Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

AGT HAL Module Clock Configuration

The AGT timer is clocked based on the PCLKB, LOCO, Fsub or AGT Underflow frequency. The AGT clock is selectable in the **Properties** window in e^2 studio. You can set the clock frequencies using the clock configurator in e^2 studio or the CGC Interface at run-time.

AGT HAL Module Pin Configuration

The AGT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins.

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the AGT HAL Module on r_agt

Resource	ISDE Tab	Pin selection Sequence
AGT	Pins	Select **Peripherals**> **Timer: AGT**> **AGT0**

Note

The selection sequence assumes AGT0 is the desired hardware target for the driver.

Pin Configuration Settings for the AGT HAL Module on r_agt

Property	Value	Description
Operation Mode	Disabled, Custom, Timer Output, Compare Match, Count Measurement, Gated Count Default: Disabled	Select timer operation mode.
AGTIO	None Default: None	AGTIO Pin.
AGTO	None, P102 Default: P102	AGTO Pin.
AGTOA	None Default: None	AGTOA Pin.
AGTOB	None Default: None	AGTOB Pin.
AGTEE	None, P101 Default: P101	AGTEE Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.5.6 Using the AGT HAL Module in an Application

The typical steps in using the AGT HAL Module in an application are:

1. Initialize the AGT HAL module using the timer_api_t::open API.
2. Start the AGT HAL module by calling the timer_api_t::start API.
3. Read the counter value by calling the timer_api_t::counterGet API.
4. Set the period value by using thetimer_api_t::periodSet API
5. Set the duty cycle by using timer_api_t::dutyCycleSet API
6. Get the timer information using timer_api_t::infoGet API
7. Respond to the AGT HAL module callback as needed.
8. Resets the counter value using the timer_api_t::reset API
9. Stop the AGT channel using timer_api_t::stop API
10. Use the timer_api_t::close call to power down the peripheral.

Note

The timer-period and duty-cycle parameters can be reconfigured based on the application's needs.

These common steps are illustrated in a typical operational flow diagram in the following figure:

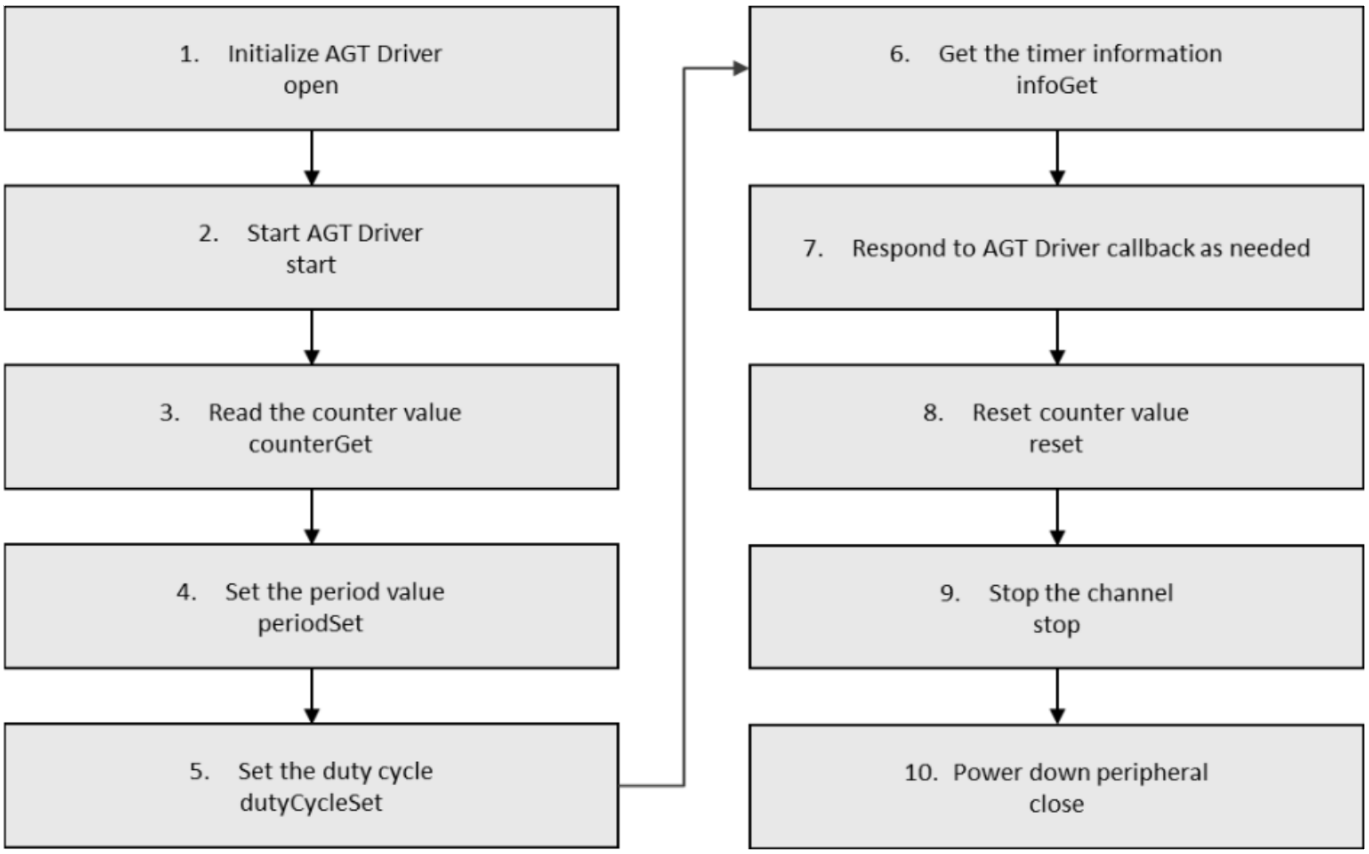


Figure 244: Flow Diagram of a Typical AGT HAL Module Application

4.2.6 AGT Input Capture Driver on r_agt

4.2.6.1 Input Capture HAL Module Introduction

The Input Capture HAL module provides an API for measuring input pulse-width, period measurement and event count measurement. The Input Capture HAL module also configures the input capture parameters to use with the AGT peripheral on the Synergy MCU. A user-defined callback can be created to acquire the value each time a new measurement is complete.

Input Capture HAL Module Features

The Input Capture HAL module configures the AGT for an input capture function.

- The Input Capture HAL allows the user to perform the following tasks:
  - Initialize the module
  - Enable input capture measurement
  - Disable input capture measurement
  - Get the status (running or not) of the measurement counter
  - Get the last captured timer/overflows counter value
  - Close the input capture operation
- The Input Capture HAL module supports:

    - Pulse-width measurement, period measurement and event count measurement
    - Rising-edge or falling-edge measurement start
    - One-shot or periodic mode
    - Callback function with the following events:
        - Measurement and overflow interrupt
        - Capture compare interrupt
- Callback structure (input_capture_callback_args_t) that provides data on the interrupting event, including which interrupt occurs and the associated counter values.

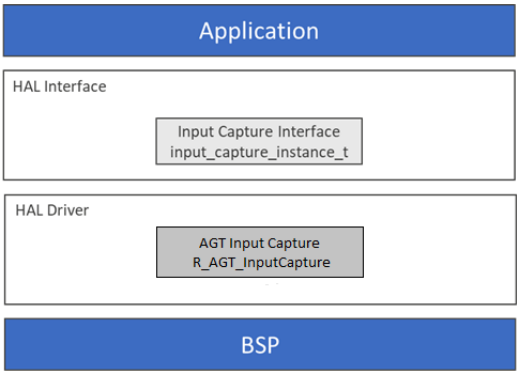


Figure 245: Input Capture HAL Module Block Diagram

AGT Input Capture Hardware support details

The following hardware features are, or are not, supported by SSP for AGT Input Capture.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Timer Mode	Pulse Output Mode	Event Counter Mode	Pulse width measurement mode
S124	☒	☒	✓	✓
S128	☒	☒	✓	✓
S1JA	☒	☒	✓	✓
S3A1	☒	☒	✓	✓
S3A3	☒	☒	✓	✓
S3A6	☒	☒	✓	✓
S3A7	☒	☒	✓	✓
S5D3	☒	☒	✓	✓
S5D5	☒	☒	✓	✓

S5D9	☒	☒	✓	✓
S7G2	☒	☒	✓	✓

MCU Group	Pulse period measurement mode	Event link function through ELC HAL driver	Compare/Match Function
S124	✓	☒	☒
S128	✓	☒	☒
S1JA	✓	☒	☒
S3A1	✓	☒	☒
S3A3	✓	☒	☒
S3A6	✓	☒	☒
S3A7	✓	☒	☒
S5D3	✓	☒	☒
S5D5	✓	☒	☒
S5D9	✓	☒	☒
S7G2	✓	☒	☒

4.2.6.2 Input Capture HAL Module APIs Overview

The Input Capture HAL module interface defines APIs for opening, closing, enabling, disabling, accessing status information, and accessing the last-capture value using the Asynchronous General-Purpose Timer (AGT) with Input Capture. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the HAL Module API Summary.

Input Capture HAL Module API Summary

Function Name	Example API Call and Description
open	g_input_capture.p_api->open(g_input_capture.p_ctrl, g_input_capture.p_cfg); Opens the Input Capture HAL and initializes configuration.
close	g_input_capture.p_api->close(g_input_capture.p_ctrl); Closes the input capture operation. Allow drive to be reconfigured, and may reduce power consumption.
enable	g_input_capture.p_api->enable(g_input_capture.p_ctrl); Enables input capture measurement.
disable	g_input_capture.p_api->disable(g_input_capture.p_ctrl); Disables input capture measurement.

infoGet	g_input_capture.p_api->infoGet(g_input_capture.p_ctrl, &input_capture_info); Gets the status (running or not) of the measurement counter.
lastCaptureGet	g_input_capture.p_api->lastCaptureGet(g_input_capture.p_ctrl, &input_capture_counter); Gets the last captured timer/overflows counter value.
versionGet	g_input_capture.p_api->versionGet(&input_capture_version); Retrieve the API version with the input_capture_version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	One of the parameters is NULL. Or the channel requested in the p_cfg parameter may not be available on the device selected in r_bsp_cfg.h. Or, p_cfg->mode is invalid.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.6.3 Input Capture HAL Module Operational Overview

The Input Capture HAL module controls the AGT HAL module units on a Synergy microcontroller (as configured by the user). It directly accesses the AGT hardware without using any RTOS elements and provides convenient APIs to simplify development.

When a normal measurement is complete and a callback is available (with interrupts enabled,) the Input Capture HAL module invokes the callback with the argument input_capture_callback_args_t.

The argument input_capture_callback_args_t indicates the channel, the event input_capture_event_t, the value of the timer captured when the interrupt occurred, and the number of counter overflows that occurred during this measurement.

If the interrupts are not enabled, the values read by the APIs would be the last captured timer/overflows counter value.

Input Capture HAL Module Important Operational Notes and Limitations

Input Capture HAL Module Operational Notes

AGT Input Capture Measurement and overflow

The input capture interface provides support for pulse-width, period measurement, and event count measurement. The input capture interface also provides support for one-shot measurement and periodic measurement. The AGT hardware does not natively support one-shot functionality. Code is automatically included in the interrupt service routine (ISR) to stop and clear the timer.

The AGT hardware doesn't have a hardware interrupt vector for event count measurement. Therefore, the compare match vector triggers a callback when the counter reaches the count value supplied by user.

AGT Input Capture Signal

The input capture measurement starts when the input capture signal edge (rising or falling) is detected on the input capture signal pin (AGTIO). The input could be captured in any one of the AGT input capture pin (such as AGTIO0_A, AGTIO0_B, or AGTIO0_C) as configured by user. The pin corresponding to this selection must also be configured in AGT peripheral in the configurator.xml (that is, under **configuration.xml > pins > peripherals > Timer:AGT).** The details on pins to be configured are in the user manual under the AGT block diagram and in the I/O ports section. Note that in the S3 and S5 series MCUs, AGTIO0_B is P402, AGTIO0_C is P403, and all other AGTIO pins are AGTIO0_A. The S1 MCU Series have only AGTIO0_A pins.

The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When three consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal.

For event count measurement, the module can be configured to count a single or both edges. That is, if it is configured as single edge, the signal edge as input by the user is counted, and if it is configured as both edges, pulses are counted on the starting edge of the pulse (either rising or falling edge). And the module will generate a callback when the pulse count reaches the event count value set by the user.

Event count measurement uses the capture compare interrupt to compare the counter value with the user expected value (and trigger a callback when they are the same), and uses the measurement and overflow interrupt to keep track of the overflow. Hence the priority of the measurement and overflow interrupt must be set higher than the latter, as the capturing of overflow takes precedence to compare match callback.

For pulse width measurement, if the input capture module is opened in the middle of an active level input by user, the module will start capturing the width from where it opened, and hence the captured width will not be the right width of the pulse.

Converting Measurement Counts to Time

When a measurement completes, the raw-count data and the number of overflows is returned to the user in the callback function.

If desired, the raw measurement data can be converted to logical time units in the callback or user

application. To convert the raw data, the current clock frequency and its pre-scaler value, number of overflows, maximum counter value, and measurement counts should be considered. The measurement counts and the number of overflows are provided in the callback arguments input_capture_callback_args_t.

The recommended method to obtain the current clock frequency is to use the cgc_api_t::systemClockFreqGet API. The input clock frequency is the clock frequency divided by the pre-scaler value and is represented as clk_freq_hz in the following Input Capture Time Calculation table. Note that pre-scaler are only available for AGTLCLK and AGTSCLK clocks

The maximum counter value for all boards is 0xFFFF. This maximum counter value plus one (since counter starts from zero) is represented as max_counts in the following table:

Input Capture Time Calculation

Desired Time Units	Formula
Nanoseconds (ns)	time_s = ((overflows * max_counts) + counter) * 1000000000 / clk_freq_hz
Microseconds (us)	time_s = ((overflows * max_counts) + counter) * 1000000 / clk_freq_hz
Milliseconds (ms)	time_s = ((overflows * max_counts) + counter) * 1000 / clk_freq_hz
Seconds (s)	time_s = ((overflows * max_counts) + counter) / clk_freq_hz

Input Capture HAL Module Limitations

- The input capture interface does not support the limited capture with Vbatt, that is, when running on battery power (which is available only for S3 MCU Series).
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.
- When the count source selected is AGTSCLK or AGTLCLK, turn on the count source using call cgc_api_t::clockStart API before calling input_capture_api_t::open.

4.2.6.4 Including the Input Capture HAL Module in an Application

This section describes how to include the Input Capture HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Input Capture Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Input Capture Driver is g_input_capture0. This name can be changed in the associated Properties window.)

Input Capture HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_input_captureInput Capture Driver on r_agt_input_capture	Threads->HAL/Common Stacks	**New Stack> Driver> Timers> Input Capture Driver on r_agt_input_capture**

When the Input Capture Driver on r_agt_input_capture is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

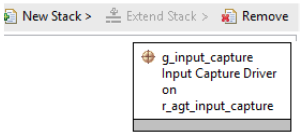


Figure 246: Input Capture HAL Module Stack

4.2.6.5 Configuring the Input Capture HAL Module

The Input Capture HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator. They are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Input Capture HAL Module on r_agt_input_capture

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_input_capture	Module name.
Channel	0	Physical hardware channel.
Mode	Pulse Width, Period, Event count Default: Pulse Width	Measures the input signal.

Signal Edge	Rising, Falling Default: Rising	Start measurement on rising or falling edge. Measurement stops on the opposite edge.
AGTIO Pin Select	AGTIO_A, AGTIO_B, AGTIO_C Default: AGTIO_A	Captures the input signal through one of these pins.
Event Edge Polarity	Single Edge, Both Edge Default: Both Edge	Counts the pulses on single edges (that is, on either rising or falling) or on both edges (that is, on either rising or falling) is received, in the event counter mode.
Repetition	Periodic, One Shot Default: Periodic	Capture a single measurement, then disable captures (one shot) until enable is called, or capture measurements continuously (periodic).
Auto Start	True, False Default: True	Set to true to enable measurements after configuring or false to leave the measurements disabled until enable is called.
Input Signal Filter	None, PCLK/1, PCLK/8, PCLK/32. Default: None	The noise filter samples the external signal at intervals of the PCLKB divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Clock source	PCLKB, PCLKB/8, PCLKB/2, AGTLCLK, AGTSCLK Default: PCLKB	Input capture clock source.
Clock Divider	PCLK/1, PCLK/2, PCLK/4, PCLK/8, PCKL/16, PCLK/32, PCLK/64, PCLK/128 Default: PCLK/1	Clock divider used to scale the measurement counter.
Callback	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) each time the measurement or overflow occurs. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Measurement and Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the measurement and overflow interrupt priority.
Capture Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the capture interrupt priority, captured interrupt is used only for event count mode.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Input Capture HAL Module Clock Configuration

The AGT HAL module uses the PCLKB, AGTLCLK and AGTSCLK as its clock source. The clock frequency is set using the SSP configurator **Clocks** tab prior to a build, or using the CGC Interface at run-time.

Input Capture HAL Module Pin Configuration

The AGT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins.

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the AGT HAL Module on r_agt

Resource	ISDE Tab	Pin selection Sequence
AGT	Pins	Select **Peripherals** > **Timer: AGT**> **AGT0**

Note

The selection sequence assumes AGT0 is the desired hardware target for the driver.

Pin Configuration Settings for the Input capture HAL Module on r_agt_input_capture

Property	Value	Description
Operation Mode	Disabled, Custom, Timer Output, Compare Match, Count Measurement, Gated Count Default: Disabled	Select timer operation mode.
AGTIO	None, P100, P402, P403 Default: None	AGTIO pin.
AGTO	None Default: None	AGTO pin.

AGTOA	None Default: None	AGTOA pin.
AGTOB	None Default: None	AGTOB pin
AGTEE	None Default: None	AGTEE pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.6.6 Using the Input Capture HAL Module in an Application

The typical steps in using the Input Capture HAL module in an application are:

1. Initialize the module using the input_capture_api_t::open API.
2. The capture and overflow interrupt can be enabled and the timer started using the input_capture_api_t::enable API.
3. The status of the captured counter (running or stopped) can be queried using input_capture_api_t::infoGet API.
4. The desired value can be found either in the main loop routine using the input_capture_api_t::lastCaptureGet API or in the callback function.
5. The capture and overflow interrupt can be disabled and the timer stopped using theinput_capture_api_t::disable API.
6. The module can be closed using the input_capture_api_t::close API once done.

These common steps are illustrated in a typical operational flow diagram in the following figure:

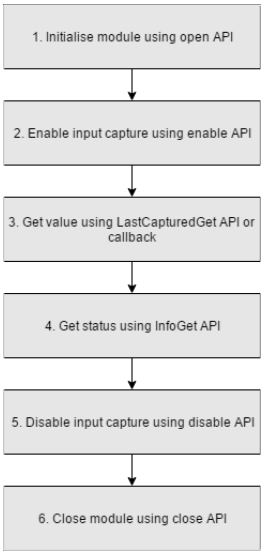


Figure 247: Flow Diagram of a Typical Input Capture HAL Module Application

4.2.7 Clock Accurate Circuit Driver

4.2.7.1 CAC HAL Module Introduction

The CAC HAL module provides a high-level API for clock accuracy control applications and uses the Clock Frequency Accuracy Measurement Circuit (CAC) peripheral on a Synergy MCU. This is particularly useful function when implementing a fail-safe mechanism for reliability-oriented applications. A user-defined callback can be created to respond to various error indications.

CAC HAL Module Features

- Supports clock frequency-measurement and monitoring based on a reference signal input.
- Reference can be either an externally supplied clock source or an internal clock sources.
- An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow.
- A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks.
- Edge-detection options for the reference clock are configurable as rising, falling, or both.

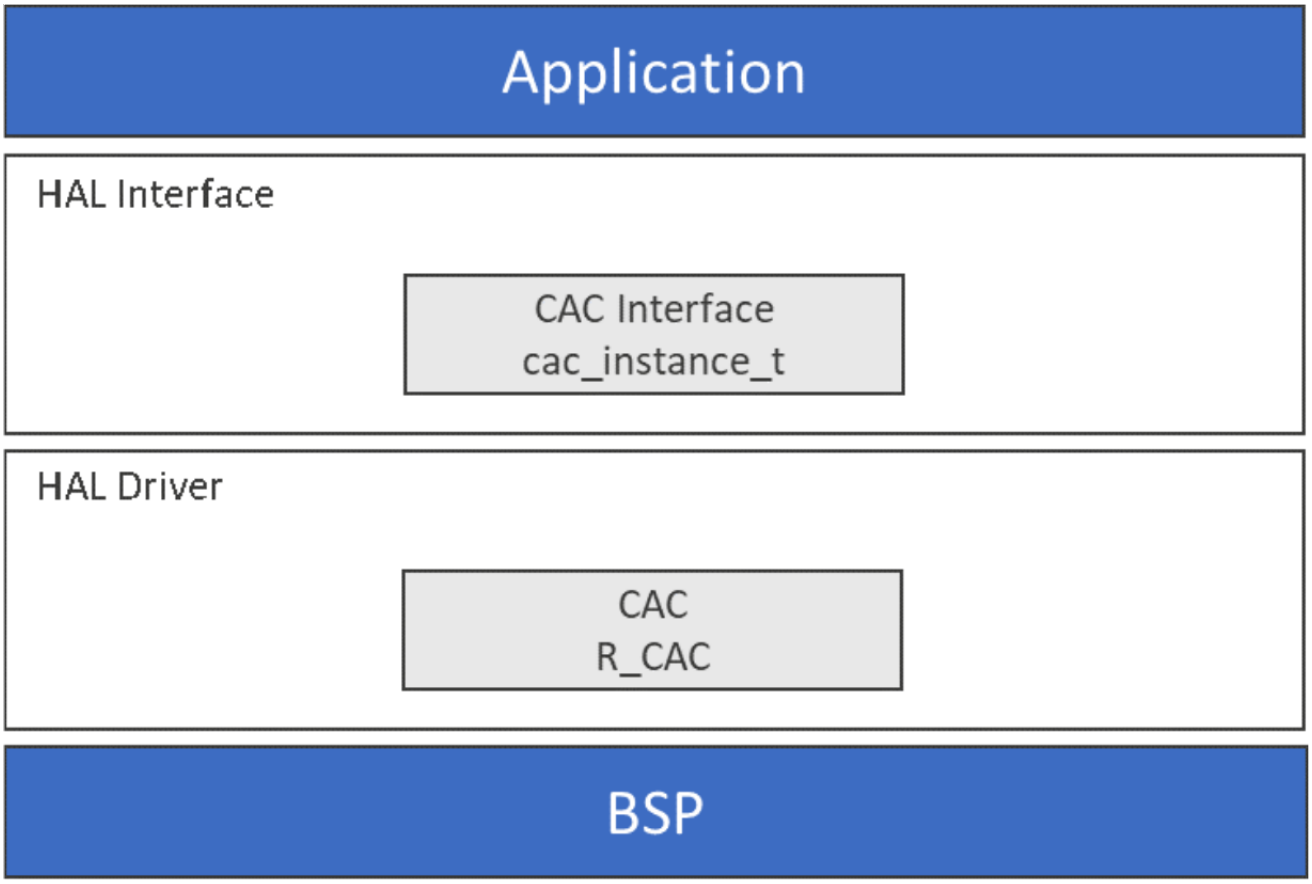


Figure 248: CAC HAL Module Block Diagram

CAC Hardware support details

The following hardware features are, or are not, supported by SSP for the CAC.

Legend:

Symbol	Meaning

✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Measurement Clock Targets: Main Clock Oscillator	Measurement Clock Targets: Sub-Clock Oscillator	Measurement Clock Targets: HOCO Clock	Measurement Clock Targets: MOCO Clock	Measurement Clock Targets: LOCO Clock
S124	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

MCU Group	Measurement Clock Targets: IWDTCLK	Measurement Clock Targets: Peripheral Module Clock B	Measurement Reference Clocks: External clock input to the CACREF pin	Measurement Reference Clocks: Main Clock Oscillator	Measurement Reference Clocks: Sub-Clock Oscillator
S124	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

MCU Group	Measurement Reference Clocks: HOCO Clock	Measurement Reference Clocks: MOCO Clock	Measurement Reference Clocks: LOCO Clock	Measurement Reference Clocks: IWDTCLK Clock	Measurement Reference Clocks: Peripheral Module Clock B
S124	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

MCU Group	Selectable Function: Digital Filter	Interrupt Sources: Measurement End	Interrupt Sources: Frequency Error	Interrupt Sources: Overflow	Module-stop function to reduce power consumption
S124	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

4.2.7.2 CAC HAL Module APIs Overview

The CAC HAL module defines APIs for opening, closing, reading, starting, stopping and resetting the CAC. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

CAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_cac0.p_api->open(g_cac0.p_ctrl, g_cac0.p_cfg) Open function for CAC device.
read	g_cac0.p_api->read(g_cac0.p_ctrl, &cac0_status, &cac0_counter) Read function for CAC peripheral.
close	g_cac0.p_api->close(g_cac0.p_ctrl) Close function for CAC device.
stopMeasurement	g_cac0.p_api->stopMeasurement(g_cac0.p_ctrl) Ends a measurement for the CAC peripheral.
startMeasurement	g_cac0.p_api->startMeasurement(g_cac0.p_ctrl) Begin a measurement for the CAC peripheral.
reset	g_cac0.p_api->reset(g_cac0.p_ctrl) Reset function for CAC device.
versionGet	g_cac0.p_api->versionGet(&cac0_version) Get the CAC API and code version information.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	One or more configuration options are invalid
SSP_ERR_NOT_OPEN	Open has not been successfully called
SSP_ERR_ASSERTION	Null provided for p_ctrl, p_cfg and others
SSP_ERR_INVALID_POINTER	Interrupt specified with NULL callback
SSP_ERR_HW_LOCKED	Hardware lock for CAC peripheral is already taken
SSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate smaller than reference clock rate

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.7.3 CAC HAL Module Operational Overview

The CAC HAL module API interfaces with a clock frequency-measurement circuit capable of monitoring the clock frequency based on a reference signal input. The reference signal may be an

externally supplied clock source or one of several available internal clock sources. An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow. A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks. Edge-detection options for the reference clock are configurable as rising, falling, or both.

The frequency of the following clocks can be measured:

- Clock output from main-clock oscillator (main clock)
- Clock output from sub-clock oscillator (sub clock)
- Clock output from high-speed on-chip oscillator (HOCO clock)
- Clock output from mid-speed on-chip oscillator (MOCO clock)
- Clock output from low-speed on-chip oscillator (LOCO clock)
- Clock output from IWDT-dedicated on-chip oscillator (IWDTCLK clock)
- Peripheral module clock (PCLKB)

The measurement clock is monitored using a reference clock. The reference clock may be an external clock (supplied on the CACREF input pin) or one of the following internal clocks:

- Clock output from main-clock oscillator (main clock)
- Clock output from sub-clock oscillator (sub clock)
- Clock output from high-speed on-chip oscillator (HOCO clock)
- Clock output from mid-speed on-chip oscillator (MOCO clock)
- Clock output from low-speed on-chip oscillator (LOCO clock)
- Clock output from IWDT-dedicated on-chip oscillator (IWDTCLK clock)
- Peripheral module clock (PCLKB)

Operational Description

The CAC HAL module measures the operation and accuracy of a selected clock. Once the measurement is requested, counting begins on the first valid edge detected for the reference clock and ends on the next valid edge. A valid edge can be configured to be rising, falling, or both. The count is incremented at each cycle of the measurement clock after it has passed through the divider circuit which is capable of dividing the clock by 1, 4, 8, or 32. An internally supplied reference clock also passes through a divider circuit which is capable of dividing the clock by 32, 128, 1024, or 8192. An externally supplied reference clock does not pass through a divider circuit, but may pass through a digital filter if it is configured to do so.

For example, if the sub-clock is specified as the measurement clock (32 kHz) and a divisor of 1 is specified, then the counter will increment at a 32 kHz rate. If a reference clock of 1 kHz is provided, then after one cycle of the reference clock you would expect the counter to be 32. This is where the CAC upper and lower-limit settings are examined. Part of the setup for a CAC measurement is the specification of an upper and lower limit for the measurement. When a measurement is complete, the CAC compares the counter contents to the limits configured for the measurement. If both the counter ≤ upper limit and the counter ≥ lower limit, then the measurement has completed without error and the measured frequency is operating within the defined limits. If the counter fails to meet these requirements, then a frequency error is indicated. A completed measurement may be identified by making API calls to poll the driver, or by establishing a callback function.

CAC HAL Module Important Operational Notes and Limitations

CAC HAL Module Operational Notes

Continuous Mode

The CAC module may be operated in either a single or continuous measurement mode. In continuous mode, the measuring process is restarted after each completed measurement. In non-continuous, or single measurement mode, measuring stops after the first completed measurement. The cac_cfg_t::continuous_mode configuration member controls this feature.

Interrupts and Callbacks

When a measurement is complete and a callback is provided by the user, (with one or more interrupts enabled) the CAC HAL module invokes the callback (cac_cfg_t::p_callback) with the argument cac_callback_args_t, indicating the event cac_event_t. If interrupts are not enabled, the API supports checking the measurement status to poll if the measurement is complete (cac_api_t::read), which will provide both the status of the measurement and the current value of the CAC counter register.

The CAC driver supports three interrupts:

- A frequency error interrupt occurs when a measurement completes and the CAC counter register value is outside of the range that was specified as part of the cac_api_t::open call. The configuration cac_cfg_t::ferr_interrupt_enabled member provided in the cac_api_t::open call must be enabled for this interrupt to be generated.
- An overflow error interrupt occurs when the CAC counter register overflows its maximum (0xFFFF) value. The configuration cac_cfg_t::ovf_interrupt_enabled member provided in the cac_api_t::open call must be enabled for this interrupt to be generated.
- A measurement complete interrupt occurs when a measurement completes and the CAC counter register value is within the range that was specified as part of the CAC driver cac_api_t::open. The configuration cac_cfg_t::mei_interrupt_enabled member provided in the cac_api_t::open call must be enabled for this interrupt to be generated.

Reset

The cac_api_t::reset API can be used to reset the overflow, measurement end and frequency error interrupt flags after measurement has been stopped to eliminate any false triggers.

CAC HAL Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.7.4 Including the CAC HAL Module in an Application

This section describes how to include the CAC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Clock Accuracy Circuit Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Clock Accuracy Circuit Driver is g_cac0. This name can be changed in the associated Properties window.)

CAC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_comparator0 Comparator Driver on r_acmphs	Threads	**New Stack> Driver> Analog> Comparator Driver on r_acmphs**

When the Clock Accuracy Circuit Driver on r_cac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

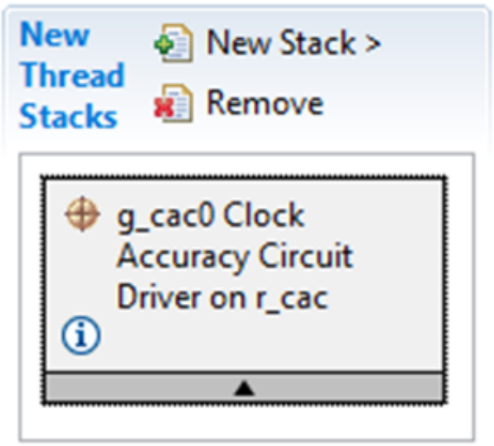


Figure 249: CAC HAL Module Stack

4.2.7.5 Configuring the CAC HAL Module

The CAC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CAC HAL Module on r_cac

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_cac0	Identifies this instance.
Continuous Measurement Operation	Enabled, Disabled Default: Enabled	If Enabled, measurement will continuously restart after completing.
Measurement Complete Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a measurement is complete, providing the CAC MEASUREMENT END interrupt is enabled in the ICU.
Overflow Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a CAC overflow occurs, providing the CAC OVERFLOW interrupt is enabled in the ICU.
Frequency Error Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a frequency error occurs, providing the CAC FREQUENCY ERROR interrupt is enabled in the ICU.
Upper Limit Threshold	0	Top end of allowable range for measurement complete.
Lower Limit Threshold	0	Bottom end of allowable range for measurement complete.
Reference Clock Source	Main Oscillator, Sub-clock, HOCO, MOCO, LOCO, PCLKB, IWDT Default: Main Oscillator	Reference clock source.
Reference Clock Divider	32,128,1024,8192 Default:32	Reference clock divider.
Reference Clock Edge Detect	Rising, Falling, Both Default: Rising	Reference clock edge detection.
Reference Clock Digital Filter	Disabled, Sampling clock = measuring frequency, Sampling clock = measuring/4, Sampling clock = measuring/16 Default: Disabled	Reference clock digital filter.

Measurement Clock Source	Main Oscillator, Sub-clock, HOCO, MOCO, LOCO, PCLKB, IWDT Default: HOCO	Measurement clock source.
Measurement Clock Divider	1,4,8,32 Default: 1	Measurement clock divider.
Callback	NULL	Function name for callback.
Frequency Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	CAC Frequency Error interrupt priority selection.
Measurement End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	CAC Measurement End interrupt priority selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	CAC Overflow interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

CAC HAL Module Clock Configuration

Clocks are selected from the **Clock** tab as needed by the application.

CAC HAL Module Pin Configuration

The pins for the CAC HAL module are selected as shown in the following table; the pin settings are shown in the subsequent table. If CACREF is used to the input pin of reference clock, you must configure this pin.

Pin Selection for the CAC HAL Module on r_cac

Resource	ISDE Tab	Pin selection Sequence
CAC HAL Module	Pins	**Peripherals> Monitoring: CAC> CAC0**

Note

The selection sequence assumes CAC0 is the desired hardware target for the driver

Pin Configuration Settings for the CAC HAL Module

Pin Configuration Property	Value	Description
Operation Mode	Disabled, External Reference Default: Disabled	Select **Enable** as the Operation Mode for CAC.
CACREF:	None, P204 Default: None	CACREF Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 kit. Other Synergy kits and MCUs may have different available pin configuration settings.

4.2.7.6 Using the CAC HAL Module in an Application

The typical steps in using the CAC HAL module in an application are:

1. Start reference clock and measurement clock using the CGC cgc_api_t::clockStart API if needed.
    a. For started clock, use the CGC cgc_api_t::clockCheck API to confirm oscillation stability or active state.
2. Get the API and code version information using the cac_api_t::versionGet if needed.
3. Initialize the CAC HAL module using the cac_api_t::open API.
4. Start a measurement using the cac_api_t::startMeasurement API.
5. Poll using the cac_api_t::read function to look for the measurement result or get measurement status and result using callback function called in ISR.
6. Stop the measurement using the cac_api_t::stopMeasurement API.
7. Reset the overflow, measurement end, and frequency error interrupt flags using the cac_api_t::reset API.
8. Close the CAC HAL module if no more measurements are needed using the cac_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

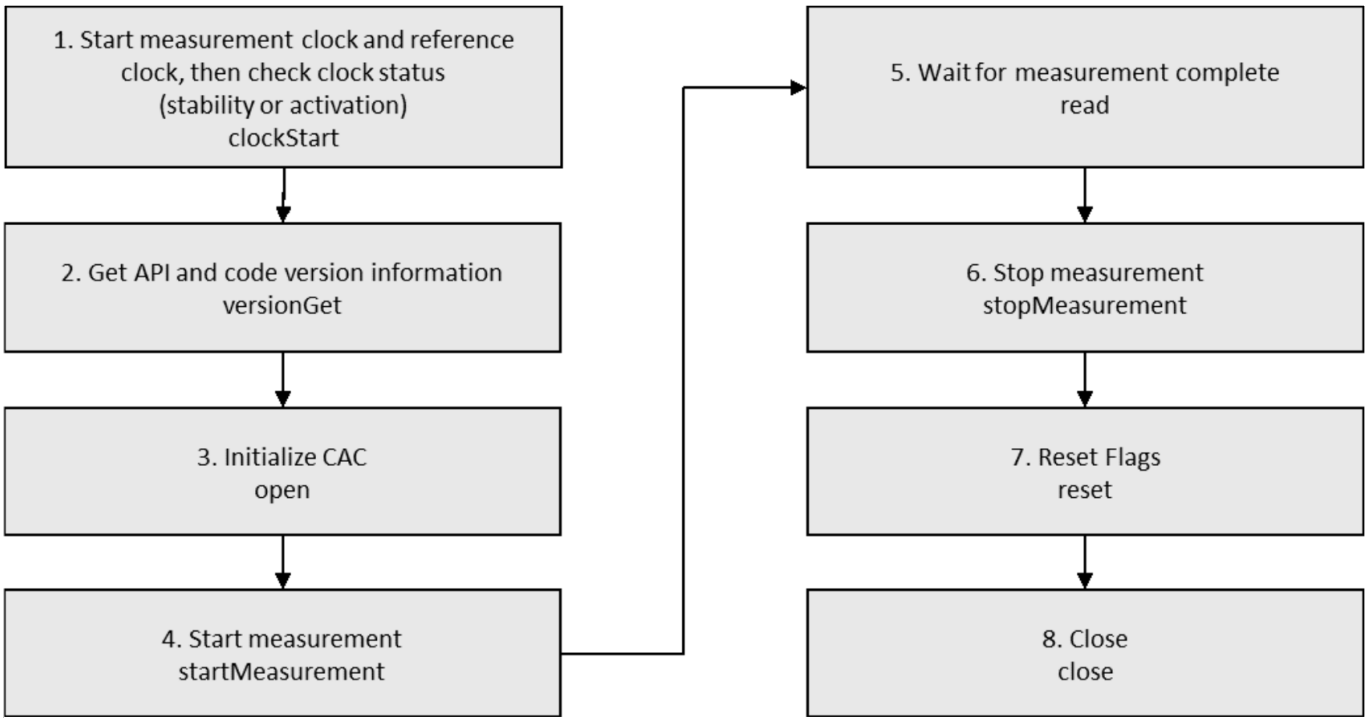


Figure 250: Flow Diagram of a Typical CAC HAL Module Application

4.2.8 CAN Driver

4.2.8.1 CAN HAL Module Introduction

The CAN HAL module provides a high-level API for CAN network applications and supports the CAN peripherals available on the Synergy microcontroller hardware. A user-callback function must be defined, which the driver will invoke when transmit, receive or error interrupts are received. The callback returns with a parameter which indicates the channel, mailbox and event.

CAN HAL Module Features

- Supports both standard (11-bit) and extended (29-bit) messaging formats
- Support for bit timing configuration as defined in the CAN specification
- Supports up to 32 transmit or receive mailboxes with standard or extended ID frames
- Receive mailboxes can be configured to capture either data or remote CAN Frames
- Receive mailboxes can be configured to receive a range of IDs using mailbox masks
- Supports a user-callback function when transmit, receive, or error interrupts are received

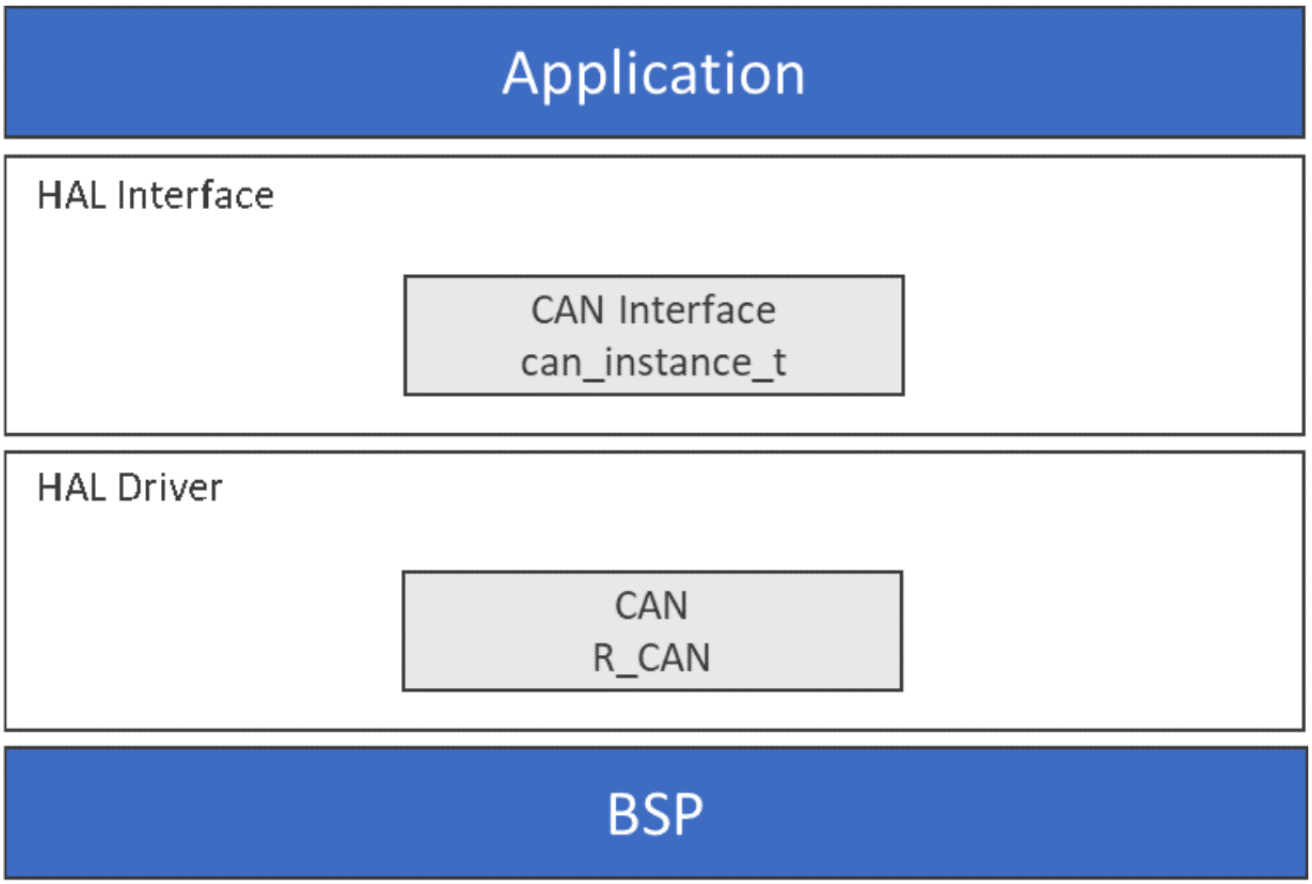


Figure 251: CAN HAL Module Block Diagram

CAN Hardware support details

The following hardware features are, or are not, supported by SSP for CAN:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Programmable Bit Rate	Support up to 32 Mailboxes	Mailbox Normal Mode	Mailbox FIFO Mode	Support for Data Frame Reception	Support for Remote Frame Reception
S124	✓	✓	✓	☒	✓	✓
S128	✓	✓	✓	☒	✓	✓
S1JA	✓	✓	✓	☒	✓	✓
S3A1	✓	✓	✓	☒	✓	✓
S3A3	✓	✓	✓	☒	✓	✓

S3A6	✓	✓	✓	☒	✓	✓
S3A7	✓	✓	✓	☒	✓	✓
S5D3	✓	✓	✓	☒	✓	✓
S5D5	✓	✓	✓	☒	✓	✓
S5D9	✓	✓	✓	☒	✓	✓
S7G2	✓	✓	✓	☒	✓	✓

MCU Group	Programmable one-shot reception	Overwrite mode reception	Overrun mode reception	Support all 8 Acceptance Masks	Support Masks independently enabled or disabled for each Mailbox
S124	☒	✓	✓	✓	✓
S128	☒	✓	✓	✓	✓
S1JA	☒	✓	✓	✓	✓
S3A1	☒	✓	✓	✓	✓
S3A3	☒	✓	✓	✓	✓
S3A6	☒	✓	✓	✓	✓
S3A7	☒	✓	✓	✓	✓
S5D3	☒	✓	✓	✓	✓
S5D5	☒	✓	✓	✓	✓
S5D9	☒	✓	✓	✓	✓
S7G2	☒	✓	✓	✓	✓

MCU Group	Support for transmission request abort	Mode transition for bus-off: ISO11898-1 specification -compliant	Mode transition for bus-off: Automatic invoking of CAN halt mode on bus-off entry	Mode transition for bus-off: Automatic invoking of CAN halt mode on bus-off end	Mode transition for bus-off: Invoking of CAN halt mode through software	Mode transition for bus-off: Transition to error active state through software.
S124	☒	✓	☒	☒	☒	☒
S128	☒	✓	☒	☒	☒	☒
S1JA	☒	✓	☒	☒	☒	☒
S3A1	☒	✓	☒	☒	☒	☒
S3A3	☒	✓	☒	☒	☒	☒
S3A6	☒	✓	☒	☒	☒	☒
S3A7	☒	✓	☒	☒	☒	☒

S5D3	☒	✓	☒	☒	☒	☒
S5D5	☒	✓	☒	☒	☒	☒
S5D9	☒	✓	☒	☒	☒	☒
S7G2	☒	✓	☒	☒	☒	☒
MCU Group	Monitoring of all CAN bus errors {Stuff, Form, ACK, 15-bit CRC, Bit error, ACK Delimiter}	Detection of all transition to error states {error warning, error passive, bus-off entry, and bus-off Recovery}	Support Interrupt Sources {Receive FIFO, Transmit FIFO }	Support Interrupt Sources {Reception complete. Transmission complete, Error interrupts}	Support CAN sleep mode 1 {stop CAN clock}	Support all 3 software units {Acceptance filter support, Mailbox search support, including receive mailbox search, transmit mailbox search, and message lost Search, Channel search support}
S124	✓	✓	☒	✓	✓	✓
S128	✓	✓	☒	✓	✓	✓
S1JA	✓	✓	☒	✓	✓	✓
S3A1	✓	✓	☒	✓	✓	✓
S3A3	✓	✓	☒	✓	✓	✓
S3A6	✓	✓	☒	✓	✓	✓
S3A7	✓	✓	☒	✓	✓	✓
S5D3	✓	✓	☒	✓	✓	✓
S5D5	✓	✓	☒	✓	✓	✓
S5D9	✓	✓	☒	✓	✓	✓
S7G2	✓	✓	☒	✓	✓	✓

MCU Group	CAN Source Clock: PCLKB	CAN Source Clock: CANMCLK	Support all 3 Test Modes {Listen-only, Self-Test 1 (external loopback), Self Test 2 (internal loopback)}	Module-stop Function	Support Standard CAN (11 bit)	Support Extended CAN (29 bit)
S124	N/A	✓	✓	✓	✓	✓
S128	N/A	✓	✓	✓	✓	✓
S1JA	N/A	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

4.2.8.2 CAN HAL Module APIs Overview

The CAN HAL defines APIs for opening, closing, writing (transmitting) and reading (receiving) CAN data; it also provides some additional functions to assist in processing more complex commands. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

CAN HAL Module API Summary

Function Name	Example API Call and Description
open	g_can0.p_api->open(g_can0.p_ctrl, g_can0.p_cfg) The open API configures CAN Channel 0. This function must be called before any other CAN functions. Note: This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the module, open will not need to be called.
close	g_can0.p_api->close(g_can0.p_ctrl) The close API handles the clean-up of internal driver data.

read	g_can0.p_api->read (g_can0.p_ctrl, p_args->mailbox, &receiveFrame) The read API reads received CAN data.
write	g_can0.p_api->write (g_can0.p_ctrl, 0, &transmitFrame) The write API write data into the CAN transmit frame buffer and send it out.

control	g_can0.p_api->control(g_can0.p_ctrl, CAN_COMMAND_MODE_SWITCH, &mode);
	withcan_mode_t mode = CAN_MODE_LOOPBACK_INTERNAL;
	The control API will be able to change the CAN mode of operation.

infoGet	g_can0.p_api->infoGet(g_can0.p_ctrl, p_info) The infoGet API retrieves the CAN mode of operation.
versionGet	g_can0.p_api->versionGet(version) The versionGet API retrieves the module version information.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_HW_LOCKED	Lock already owned by another user.
SSP_ERR_CAN_MODE_SWITCH_FAILED	Channel failed to switch modes.
SSP_ERR_CAN_INIT_FAILED	Channel failed to initialize.
SSP_ERR_ASSERTION	Null pointer presented.
SSP_ERR_NOT_OPEN	Port is not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	No data available.
SSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is not setup for receive.
SSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
SSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup for receive and cannot send.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.8.3 CAN HAL Module Operational Overview

The CAN HAL module controls the CAN peripherals on Synergy microcontrollers according to the user configuration. The API provides open, close, read, write, control and information functions. The driver allows for bit-timing configuration as defined in the CAN specification; it can be configured for up to 32 transmit or receive mailboxes with standard or extended ID frames. Receive mailboxes can be configured to capture either data or remote CAN frames. The user callback is invoked with the channel, mailbox and event information when a transmit, receive or error interrupt occurs.

Using the CAN IDs and Masks

Each CAN Mailbox configured to receive messages has an ID and Mask set. Incoming messages will be placed in the lowest mailbox where the following is true:

Incoming ID & Mailbox Mask == Mailbox ID & Mailbox Mask

Example 1: A mailbox with an ID of 0x25 and a mask of 0x1FFFFFFF can receive messages with IDs of 0x25.

Example 2: A mailbox with an ID of 0x25 and a mask of 0x1FFFFFF0 can receive messages with IDs of 0x20 through 0x2F.

Using the CAN HAL Module Test Modes

The CAN Module has three test modes, listen only, external loopback and internal loopback.

- In the listen only mode valid data frames and remote frames can be received. However, only recessive bits can be sent on the CAN bus. The ACK bit, overload flag, and active error flag cannot be sent. Listen only mode can be used for baud rate detection
- In the external loopback mode the protocol module treats its own transmitted messages as those received by the CAN transceiver and stores them into the receive mailbox. To be independent from external stimulation, the protocol module generates the ACK bit. Connect the CTX and CRX pins to the transceiver.
- The internal loopback mode is similar to the external loopback mode except the protocol controller performs internal loopback from the internal CTX pin to the internal CRX pin. The input value of the external CRX pin is ignored. The external CTX pin outputs only recessive bits. The CTX and CRX pins are not required to be connected to the CAN bus or any external device.

Changing the CAN HAL Module Operating Modes

The CAN Module can be switched between modes using the can_api_t::control API. Pass CAN_COMMAND_MODE_SWITCH and a pointer to a can_mode_t variable set to the desired mode into the can_api_t::control API.

Mode	can_mode_t value	Reason for use
Normal	CAN_MODE_NORMAL	Normal operation mode
Internal Loopback	CAN_MODE_LOOPBACK_INTERNAL	Internal loopback testing
External Loopback	CAN_MODE_LOOPBACK_EXTERNAL	External loopback testing
Listen Only	CAN_MODE_LISTEN	Baud rate detection

Halt	CAN_MODE_HALT	Mailbox configuration and test mode setting
Sleep	CAN_MODE_SLEEP	Stops the clock supply to the CAN module reducing current consumption
Exit Sleep	CAN_MODE_EXIT_SLEEP	Internal use only
Reset	CAN_MODE_RESET	Communication configuration

Example usage:

```
    /* Switch the device into internal loopback mode for easy testing. */
    can_mode_t mode = CAN_MODE_LOOPBACK_INTERNAL;
    ssp_err_t error = g_can0.p_api->control(g_can0.p_ctrl, CAN_COMMAND_MODE_SWITCH,
&mode);
```

CAN HAL Module Important Operational Notes and Limitations

CAN HAL Module Operational Notes

- The user application must start the main-clock oscillator (CANMCLK or XTAL) at run-time using the CGC Interface if it has not already started (for example, if it is not used as the MCU clock source.)
- For S7, S5, S3 and S1 MCUs, the following clock restriction must be satisfied for the CAN HAL module when the clock source is the main-clock oscillator (CANMCLK).fPCLKB >= fCANCLK (XTAL / Baud Rate Prescaler)
- For S7, S5 and S3 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module when the clock source is PCLKB.
- For S3 MCUs, the clock frequency ratio of PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For S1 MCUs, the clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- SJW (Synchronization Jump Width) is often given by the bus administrator. Select 1 <= SJW <= 4.
- Time segment and SJW settings must adhere to the following constraints: TS1 > TS2 >= SJW.

CAN HAL Module Limitations

- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.8.4 Including the CAN HAL Module in an Application

This section describes how to include the CAN HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of

the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the CAN Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the CAN Driver is g_can0. This name can be changed in the associated Properties window.)

CAN HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_can0 CAN Driver on r_can	Threads	**New Stack> Driver> Connectivity> CAN Driver on r_can**

When the CAN Driver on r_can is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

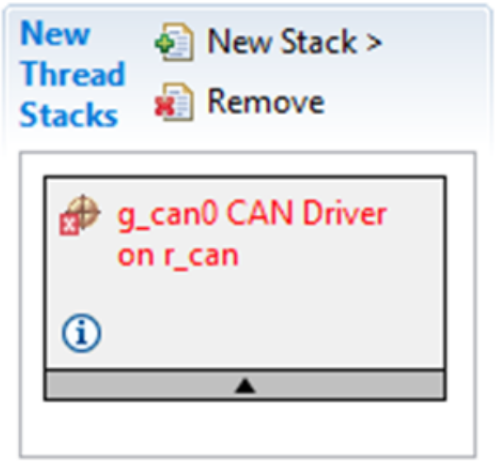


Figure 252: CAN HAL Module Stack

4.2.8.5 Configuring the CAN HAL Module

The CAN HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CAN HAL Module on r_can

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_can0	Specify CAN Module instance name.
Channel	0	Specify if the CAN channel is to use 0 or 1 (S7G2 only).
Baud Rate Prescaler	5	Specify the baud rate prescaler (0-1023). See CAN Bit Rate Calculation.
Time Segment 1	4 Time Quanta thru 16 Time Quanta Default: 15 Time Quanta	Specify the time segment 1 value (4-16). See CAN Bit Rate Calculation.
Time Segment 2	2 Time Quanta thru 8 Time Quanta Default: 8 Time Quanta	Specify the time segment 2 value (2-8). See CAN Bit Rate Calculation.
Synchronization Jump Width	1 Time Quanta thru 4 Time Quanta Default: 2 Time Quanta	Specify the Synchronization Jump Width value (1-4). See CAN Bit Rate Calculation.
Clock Source	PCLKB (S7G2 and S3A7 only), CAN MCLK Default: CAN MCLK	CAN clock source, CANMCLK or PCLKB (S7G2 and S3A7 only).
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Overwrite/Overrun Mode	Overwrite Mode, Overrun Mode Default: Overwrite Mode	Select whether receive mailbox will be overwritten or overrun if data is not read in time.
Standard or Extended ID Mode	Standard ID Mode, Extended ID Mode Default: Standard ID Mode	Select whether the driver will be using CAN standard or extended IDs.
Number of Mailboxes	4, 8, 16, 32 Mailboxes Default: 32 Mailboxes	Select 4, 8, 16 or 32 mailboxes.
Mailbox 0-31 ID	0-31	Select the receive ID for mailbox 0, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Mailbox 0 Type	Receive Mailbox, Transmit Mailbox Default: Transmit Mailbox	Select whether the mailbox is used for receive or transmit.
Mailbox 1-31 Type	Receive Mailbox, Transmit Mailbox Default: Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Mailbox 0 Frame Type	Data Mailbox, Remote Mailbox Default: Remote Mailbox	Select whether the mailbox is used to capture data frames or remote frames (receive only).
Mailbox 1-31 Frame Type	Data Mailbox, Remote Mailbox Default: Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (receive only).
Mailbox 0-3 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 0-3. See Setting the Mailbox Group Masks.
Mailbox 4-7 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 4-7. See Setting the Mailbox Group Masks.
Mailbox 8-11 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 8-11. See Setting the Mailbox Group Masks.
Mailbox 12-15 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 12-15. See Setting the Mailbox Group Masks.
Mailbox 16-19 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 16-19. See Setting the Mailbox Group Masks.

Mailbox 20-23 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 20-23. See Setting the Mailbox Group Masks.
Mailbox 24-27 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 24-27. See Setting the Mailbox Group Masks.
Mailbox 28-31 Group Mask	0x1FFFFFFF	Select the Mask for mailboxes 28-31. See Setting the Mailbox Group Masks.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Specify the error interrupt priority 0-15 (required).
Receive Mailbox Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Specify the receive interrupt priority 0-15 (required).
Transmit Mailbox Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Specify the transmit interrupt priority 0-15 (required).

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

CAN HAL Module Clock Configuration

The CAN peripheral uses the CANMCLK (main-clock oscillator) or PCLKB (S7G2, S5D9, S5D5, S3A7 and S3A7 only) as its clock source (fCAN, CAN System Clock.) Using the PCLKB with the default of 60 MHz and the Synergy default, (S7G2 DK) CAN configuration will provide a CAN bit rate of 500 Kbit.

To set the PCLKB frequency, use the clock configurator in e^2 studio. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

CAN HAL Module Pin Configuration

The CAN peripheral module uses the pins on the MCU to communicate to external devices connected on the CAN bus. Under Peripherals, select CAN and then CAN0 for channel 0 or CAN1 (S7G2 and S3A7 only) for channel 1. The operation mode for the channel must be enabled and the CRXn and CTXn pins must be selected to match your PC board layout. The pin configurator sets appropriate CAN pin configuration in the pin_cfg field for the associated pin. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the CAN pins.

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the CAN HAL Module on r_ can

Resource	ISDE Tab	Pin selection Sequence
CAN	Pins	Select **Peripherals**> **CAN**> **CAN0**

Note

The selection sequence assumes CAN0 is the desired hardware target for the driver.

Pin Configuration Settings for the CAN HAL Module on r_ can

Property	Value	Description
Operation Mode	Disabled, Enabled	Enable the mode to use CAN0.
CRX	None, P202, P402 Default: P402	CAN0_CRX0.
CTX	None, P203 P401 Default: P401	CAN0_CTX0.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

CAN Bit Rate Calculation

A time quanta (Tq) is one bit-time of the CAN communication clock, fCANCLK. This is not the CAN bit-time but the internal clock period of the CAN peripheral. The frequency is determined by the baud-rate prescaler value and the CAN source clock, fCAN (CANMCLK or PCLKB). One bit-time is divided into a number of time quanta, Tqtot. One time quantum is equal to the period of the fCANCLK. Each bitrate register is then given a certain number of Tq of the total of Tq that make up one CAN-bit period. The default ISDE bitrate setting (S7G2 DK template) is 500 Kbps for a fCAN at 60 MHz (using PCLKB.)

The formulas to calculate the bitrate register settings are as follows:

fCAN = (fPCLKB or fCANMCLK)

The baud-rate prescaler scales the CAN peripheral clock down.

fCANCLK = fCAN/ Baud Rate Prescaler = 60 MHz (default)/ 5(default) = 12 MHz

One time quantum is one clock period of the CAN clock.

Tqtot =1/fCANCLK

Tqtot is the total number of CAN peripheral clock cycles during one CAN bit time and is by the peripheral built by the sum of the "time segments" and "SS" which is always 1.

Tqtot = TSEG1 + TSEG2 + SS (TSEG1 must be > TSEG2) = 15 + 8 + 1 = 24 (default)

The bitrate is then:

Bitrate = fCANCLK/Tqtot = 12 MHz / 24 = 500 Kbps

Important notes:

- The user application must start the main-clock oscillator (CANMCLK or XTAL) at run-time using the CGC Interface if it is not already started (for example, if it is not used as the MCU clock source.)
- For S7G2, S3A7 and S124 MCUs, the following clock restriction must be satisfied for the CAN module when the clock source is the main-clock oscillator (CANMCLK): fPCLKB >= fCANMCLK
- For S7G2 and S3A7 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module when the clock source is PCLKB.
- For S3A7 MCUs, the clock frequency ratio, PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For S124 MCUs, the clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- SJW (Synchronization Jump Width) is often given by the bus administrator. Select 1 <= SJW <= 4.

Configurator sample values for different CAN bit-rate

fCAN (either PCLKB or CAN MCLK)	Baud Rate Prescalar	fCANCLK = fCAN / Baud Rate Prescalar	Time Segment 1 (TSEG1)	Time Segment 2 (TSEG2)	Synchronization Jump Width (SS)	Tqtot = TSEG1 + TSEG2 +SS	Bitrate = fCA NCLK / Tqtot
240	10	24	15	8	1	24	1 Mbps
60	5	12	15	8	1	24	500 kbps
240	48	5	16	2	2	20	250 kbps
240	96	2.5	16	2	2	20	125 kbps

Setting the Mailbox Group Masks

There are 8 mailbox group-masks, one for each group of 4 mailboxes. These masks allow the mailboxes to be configured to receive more than one ID. If the mask is all ones (0x7ff for standard IDs or 0x1FFFFFFF for extended ID) the mailboxes within the group do not mask any bits of the ID, requiring all bits of the mailbox ID to match the mailbox ID before a message is captured. If any bits of the mask are set to zero, those bits will not be necessary to match the same bits of the mailbox ID. For example, if Mailbox ID 1 is set to 0x7ff and Mailbox 0-3 Group Mask is set to 0x7ff, mailbox 1 will only capture messages with the ID of 0x7ff. If the mailbox 0-3 group mask is set to 0x7fe, mailbox 1 will still capture messages with IDs of 0x7f but will also capture messages with IDs of 0x7fe.

4.2.8.6 Using the CAN HAL Module in an Application

A CAN application requires a minimum of two nodes to demonstrate CAN communication. One node can be a transmitter, while the other can be a receiver (or both can behave as transmitter and receiver.)

The typical steps in using the CAN HAL Module in an application are:

1. Initialize the CAN HAL Module using the can_api_t::open API.

2. (Optional) Enter internal loopback or external loopback test modes using can_api_t::control API.
3. (Optional) Information about the module status, including bit rate, can be retrieved using the can_api_t::infoGet API.
4. To transmit a message:
    a. Create and configure the CAN frame ensuring correct ID and frame type.
    b. Write the CAN frame to a mailbox configured in transmit mode using the can_api_t::write API.
5. To receive a message:
    a. Read from a mailbox that has received a frame using the can_api_t::read API.
6. Close the CAN HAL Module using the can_api_t::close API (if needed).

These common steps are illustrated in a typical operational flow diagram in the following figure:

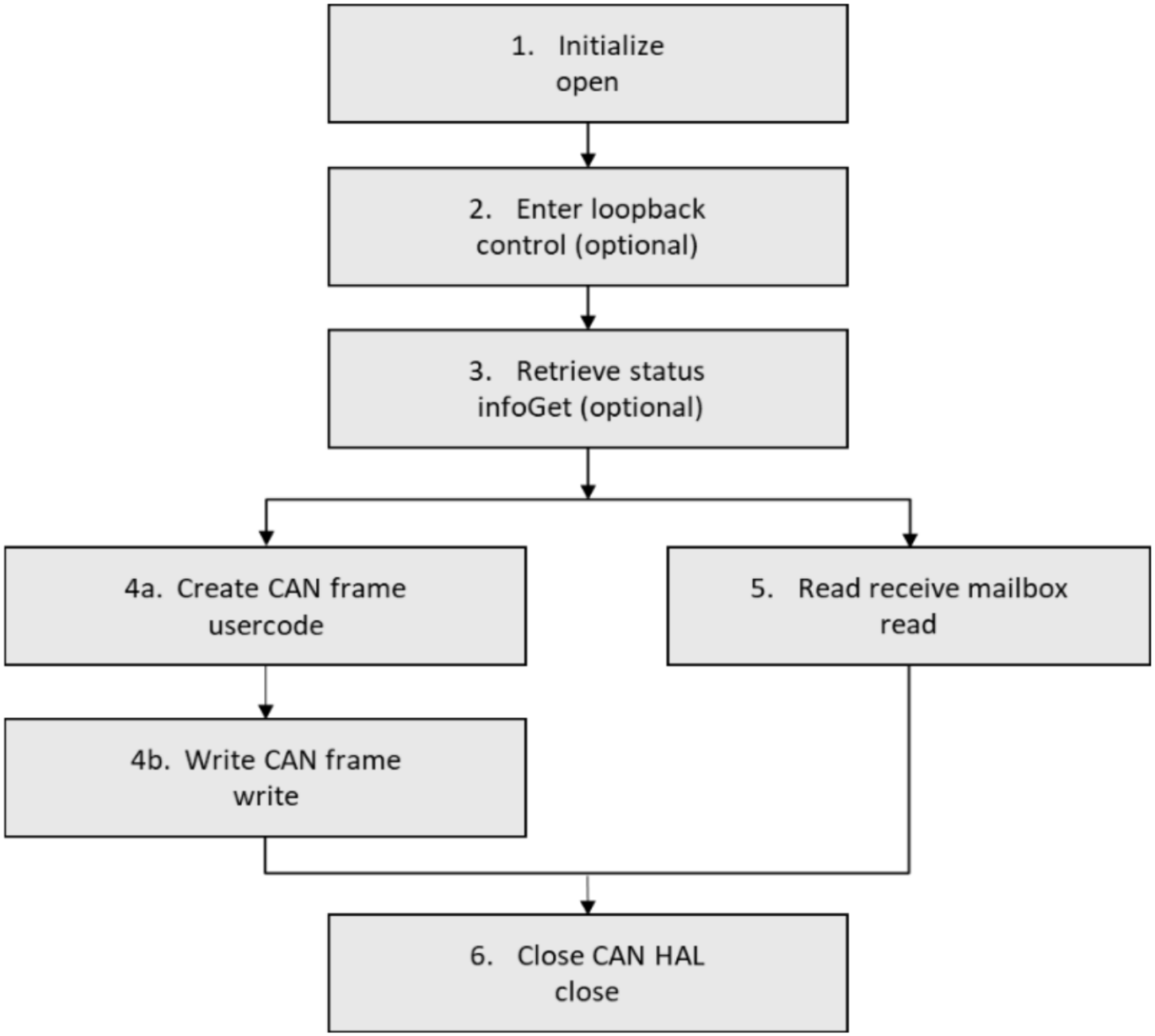


Figure 253: Flow Diagram of a Typical CAN HAL Module Application

4.2.9 CGC Driver

4.2.9.1 CGC HAL Module Introduction

The CGC HAL module provides a high-level API for clock-control applications and configures and controls the clock-control functions of a Synergy MCU using the clock-control peripheral. Since every project requires a clock function, the CGC HAL module is added to a project by default. (The module is configured in the ISDE.) A user-defined callback can be created to signal when the main oscillator has stopped.

CGC HAL Module Features

The CGC HAL module supports the configuration and control of the various clocking functions on the Synergy MCU. Key features include the following:

- Selects the system clock source

• HOCO (high-speed on-chip oscillator), MOCO (middle-speed on-chip oscillator), LOCO (low-speed on-chip oscillator), Main Clock, PLL, or Sub-Oscillator

- Configures internal clocks and turns them on or off
- Configures the output clocks
- Sets up the Oscillation Stop Detection feature
- Sets up clock divisors on each of the up to six clock domains
- Some Synergy MCUs also support controllable external clock outputs, which may have independent divisors

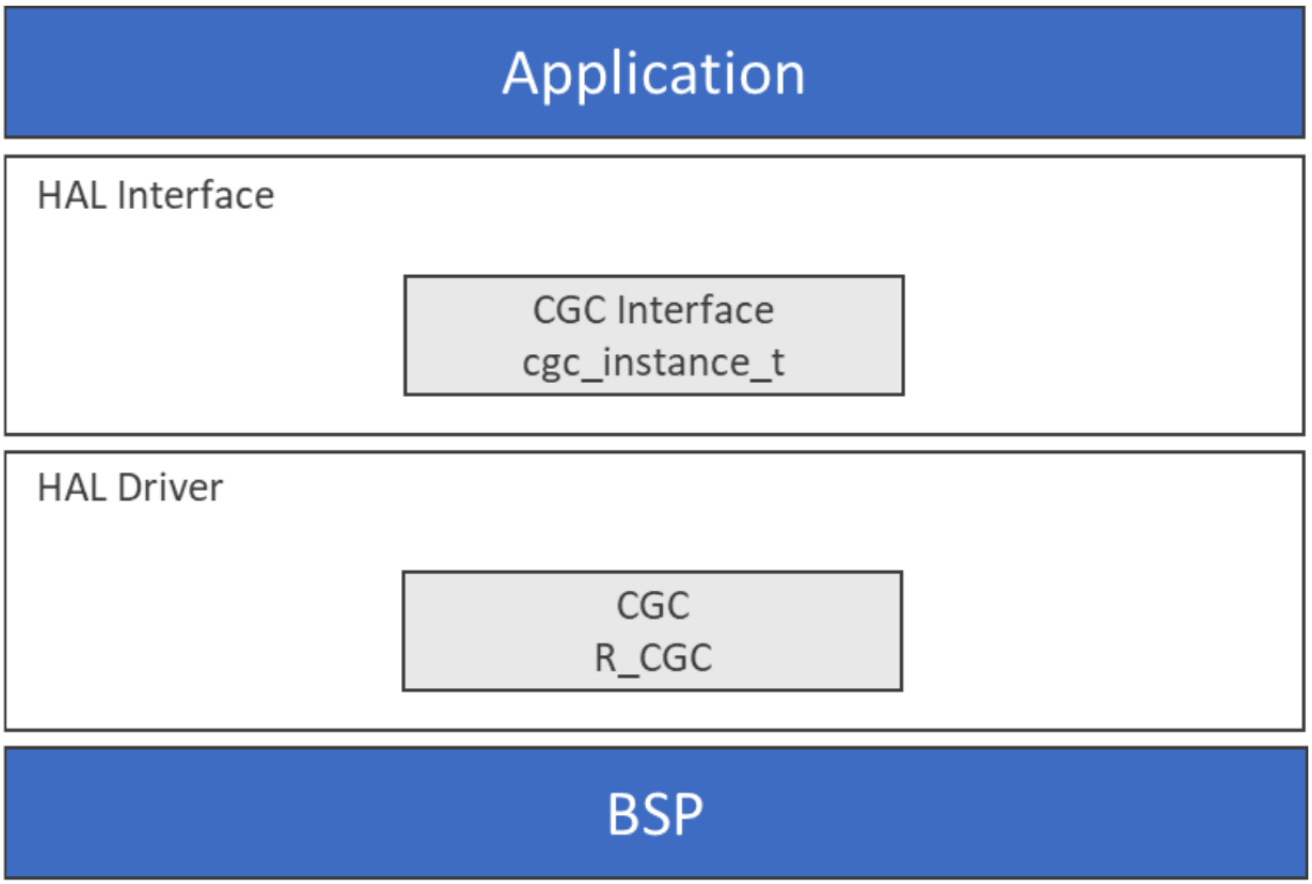


Figure 254: CGC HAL Module Block Diagram

CGC Hardware support details

The following hardware features are, or are not, supported by SSP for the Clock Generation Circuit specifications for the clock sources.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	MOSC	SOSC	PLL Circuit	HOCO	MOCO	LOCO
S124	✓	✓	N/A	✓	✓	✓
S128	✓	✓	N/A	✓	✓	✓
S1JA	✓	✓	N/A	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓

S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	IWDTLOCO	JTAG External clock input	SWD External clock input	Low Voltage Operating Power Control Mode
S124	✓	N/A	☒	✓
S128	✓	N/A	☒	✓
S1JA	✓	N/A	☒	✓
S3A1	✓	✓	✓	✓
S3A3	✓	✓	✓	✓
S3A6	✓	✓	✓	✓
S3A7	✓	✓	✓	✓
S5D3	✓	✓	✓	N/A
S5D5	✓	✓	✓	N/A
S5D9	✓	✓	✓	N/A
S7G2	✓	✓	✓	N/A

4.2.9.2 CGC HAL Module APIs Overview

The CGC HAL module defines APIs for initializing, starting, controlling and stopping the MCU clock. A complete list of the available APIs, an example API call and a short description of each can be found in the following API Summary table. A table of status return values are listed after the API summary.

CGC HAL Module API Summary

Function Name	Example API Call and Description
init	g_cgc.p_api->init(); Initial clock configuration called by BSP automatically.
clocksCfg	g_cgc.p_api->clocksCfg(&p_clock_cfg); The BSP calls this function at startup, but it can also be called from the application to change clocks at runtime.
clockStart	g_cgc.p_api->clockStart(clock_source, &p_clock_cfg); Start a clock.

clockStop	g_cgc.p_api->clockStop(clock_source); Stop a clock.
systemClockSet	g_cgc.p_api->systemClockSet(clock_source, &p_clock_cfg); Set the system clock.
systemClockGet	g_cgc.p_api->systemClockGet(&clock_source, &clock_config); Get the system clock information.
systemClockFreqGet	g_cgc.p_api->systemClockFreqGet(&clock_source, &frequency_hz); Return the frequency of the selected clock.
clockCheck	g_cgc.p_api->clockCheck(clock_source); Check the stability of the selected clock.
oscStopDetect	g_cgc.p_api->oscStopDetect(callback, enable); Configure the Main Oscillator stop detection.
oscStopStatusClear	g_cgc.p_api->oscStopStatusClear(); Clear the oscillator stop detection flag.
busClockOutCfg	g_cgc.p_api->busClockOutCfg (divider); Configure the bus clock output secondary divider. The primary divider is set using the BSP clock configuration and the systemClockSet function (S7G2 and S3A7 MCU only).
busClockOutEnable	g_cgc.p_api->busClockOutEnable (); Enable the bus clock output (S7G2 and S3A7 MCU only).
busClockOutDisable	g_cgc.p_api->busClockOutDisable (); Disable the bus clock output (S7G2 and S3A7 MCU only).
clockOutCfg	g_cgc.p_api->clockOutCfg(clock_source, clock_dividers); Configure clockOut.
clockOutEnable	g_cgc.p_api->clockOutEnable(); Enable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
clockOutDisable	g_cgc.p_api->clockOutDisable(); Disable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
lcdClockCfg	g_cgc.p_api->lcdClockCfg(clock); Configure the segment LCD Clock (S3A7 and S124 MCUs only).
lcdClockEnable	g_cgc.p_api->lcdClockEnable(); Enable the LCD clock (S3A7 and S124 MCUs only).

lcdClockDisable	g_cgc.p_api->lcdClockDisable(); Disables the LCD clock (S3A7 and S124 MCUs only).
sdadcClockCfg	g_cgc.p_api->sdadcClockCfg(clock); Configure the source for the SDADCCLK (S1JA only).
cgc_api_t::sdadcClockEnable	g_cgc.p_api->sdadcClockEnable(); Enable the SDADCCLK output (S1JA only)
cgc_api_t::sdadcClockDisable	g_cgc.p_api->sdadcClockDisable(); Disable the SDADCCLK output (S1JA only)
sdramClockOutEnable	g_cgc.p_api->sdramClockOutEnable(); Enables the SDRAM clock output (S7G2 MCU only).
sdramClockOutDisable	g_cgc.p_api->sdramClockOutDisable(); Disables the SDRAM clock (S7G2 only).
usbClockCfg	g_cgc.p_api->usbClockCfg(divider); Configures the USB clock (S7G2 only).
systickUpdate	g_cgc.p_api->systickUpdate(period_count, units); Update the Systick timer.
versionGet	g_cgc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ABORTED	Attempt to update systick timer failed.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.
SSP_ERR_STABILIZED	Clock stabilized.
SSP_ERR_CLOCK_INACTIVE	Clock not turned on.
SSP_ERR_MAIN_OCO_INACTIVE	Main OCO off/unstable.
SSP_ERR_CLOCK_ACTIVE	Clock active.
SSP_ERR_NOT_STABILIZED	Clock source un-stabilized.
SSP_ERR_CLKOUT_EXCEEDED	Clock out exceeded.
SSP_ERR_NULL_PTR	Pointer null.
SSP_ERR_OSC_DET_ENABLED	Oscillation stop detection enabled.

SSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.
SSP_ERR_INVALID_ARGUMENT	Invalid argument.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.9.3 CGC HAL Module Operational Overview

The CGC HAL module interface provides the ability to configure and use all of the CGC HAL module's capabilities. Among those capabilities are the selection of several clock sources to use as the system clock source; additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

The CGC HAL module can be used to:

- Configure any of the available clocks (HOCO, MOCO, LOCO, Main Clock, PLL, Sub-Oscillator) as the system clock source
- Configure the internal clocks (ICLK, PCLK, and so on)
- Switch the clocks on and off
- Configure the output clocks
- Set up the Oscillation Stop Detection feature

The Clock Generation Circuit peripheral features the following oscillators and clock generators:

- Main oscillator input of up to 24 MHz
- A 32.768 kHz sub-clock oscillator
- HOCO running at up to 64 MHz (depending on the device version)
- MOCO running at 8 MHz
- LOCO running at 32.768 kHz
- PLL circuit output running between 24 MHz and 240 MHz, depending on the device

The Synergy microcontrollers have six internal clock domains. Each of them has independent divisors but are dependent upon the clock input selected in the System Clock Control Register. These are:

- ICLK – The core clock, for CPU, DMAC, ROM and RAM (max 32/48/240 MHz)
- PCLKA – Peripheral clock for modules including EtherC, EDMAC, USB2.0 HS, QSPI and SCIF (max 32/48/120 MHz)

- PCLKB – Peripheral clock for modules like IIC, CAN, DAC12, RTC, USBFS, I/O Ports, WDT and IWDT (max 32/60 MHz)
- PCLKC – Peripheral clock for ADC12 conversion clock (max 64/60 MHz)
- PCLKD – Peripheral clock for GPT count clock (max 64/120 MHz)
- FCLK –Clock source for the flash memory (max 32/60 MHz)

In addition, some of the Synergy microcontrollers also support controllable external clock outputs, some of which have independent divisors. These are:

- CLKOUT – CLOCKOUT/BUZZER clock (max 24 MHz) (independent clock selector and divisor)
- BCLK – External bus clock to external bus controller (max 16/120 MHz)
- SDCLK – SDRAM clock (max 120 MHz)
- UCLK – USB clock (max 120 MHz) (independent divisor/selector on Synergy version 2)
- LCD_CLK – LCD Clock (independent clock selector but no divisor)

CGC HAL Module Changing the System Clock Peripheral Clock Divisors at Runtime

The CGC HAL module also has the option to change the system clock and clock tree settings at runtime via the cgc_api_t::clocksCfg API function. Choose **New Stack** > **System** > **CGC Configuration Instance** to create a configuration structure for use with the cgc_api_t::clocksCfg API function.

The cgc_api_t::clocksCfg function allows changes to the system clock, the peripheral clock dividers, the PLL multiplier and divider, and the state of the system clocks (stop/start) (HOCO, Main Oscillator, Subclock oscillator, and so on). The options in the following figure show an example where the system clock is being changed from HOCO to MOCO, and the peripheral clock dividers are also being updated. The options for each clock are Start, Stop, and None (meaning no change.) Not all clocks are available on all MCUs. Not all peripheral clocks are available on all MCUs.

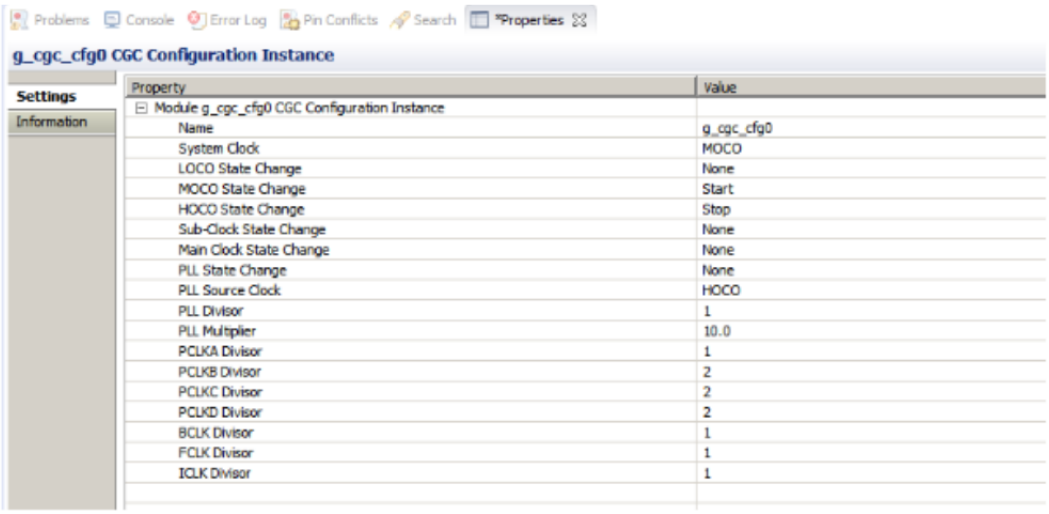


Figure 255: CGC Configuration Properties

The function call for the above example is:

```
g_cgc.p_api->clocksCfg(&g_cgc_cfg0);
```

CGC HAL Module Option Setting Memory

Synergy microcontrollers all include an Option-Setting Memory, this memory can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO. The following table lists CGC HOCO parameters that can be configured by OFS registers.

OFS register setting possibilities

Control	Description
HOCO oscillation enable	Automatically starts the HOCO after a Reset, if enabled.
HOCO Frequency	S7 and S5 Series - 16 MHz - 18 MHz - 20 MHz S3 and S1 Series - 24 MHz - 32 MHz - 48 MHz - 64 MHz

You can set the OFS register values through the properties dialog, the properties dialog is available on the Synergy Configuration editor when you select the **BSP** tab.

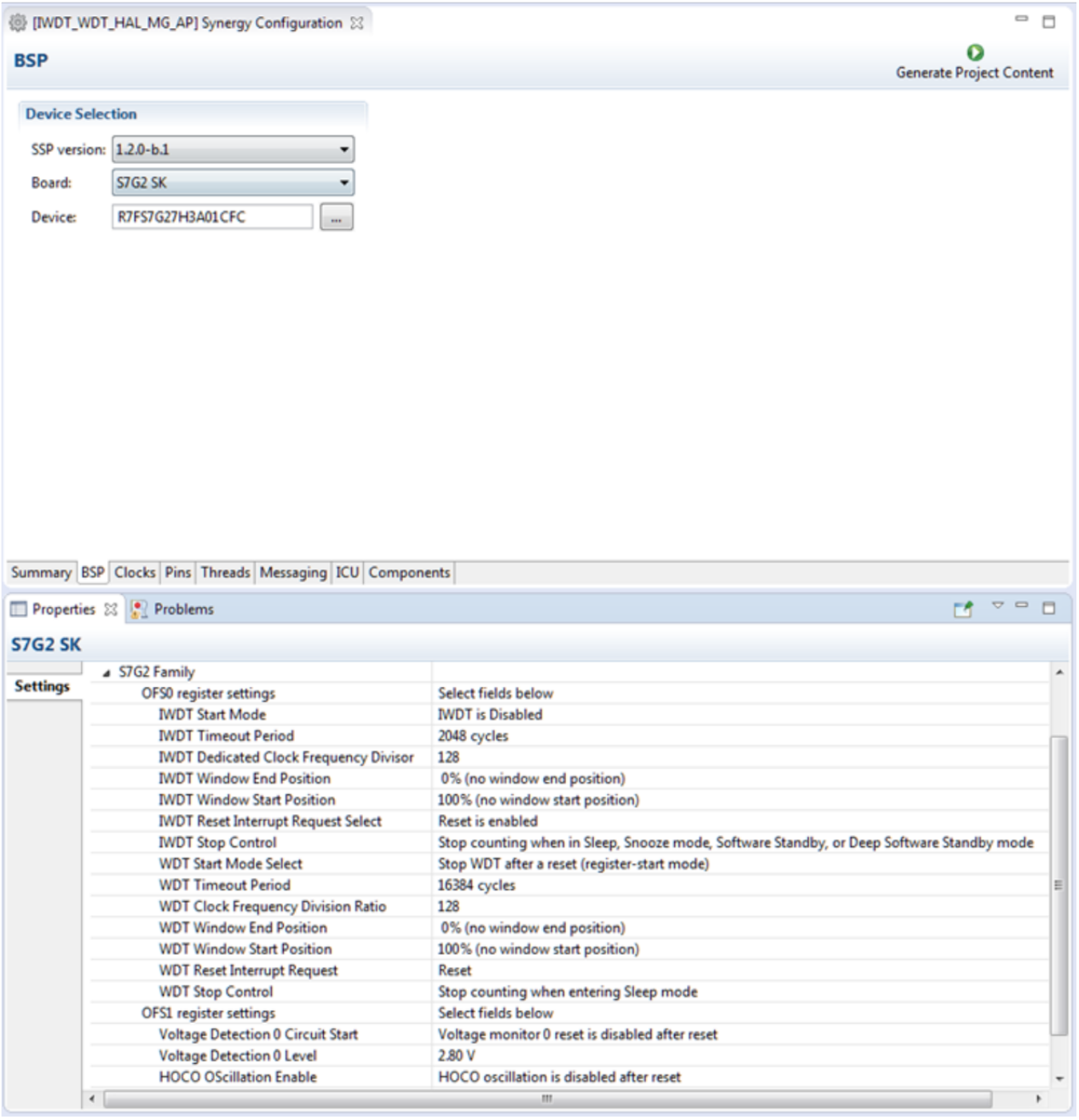


Figure 256: OFS Register Settings

Low Power Operation

The CGC HAL module also handles operating power control modes of the MCU since the Low Power Modes Version 2 HAL module will no longer handle operation-power control modes of the MCU.

When the operating power control mode is set to Sub-Oscillator Speed Mode, all system clocks except the Sub-Oscillator and the LOCO clock are turned off. To further lower the power consumption, the application program can call the clockStop API function to turn off the LOCO clock.

CGC HAL Module Important Operational Notes and Limitations

CGC HAL Module Operational Notes

- The CGC HAL module has no dependencies with respect to the ThreadX RTOS.
- The CGC HAL module is a core function of the MCU and is set after by the BSP initialization process. It is quite possible that the CGC can be left unchanged. However, the CGC HAL module provides functions that change the clock configuration that can balance the requirements of operating speed and power consumption, depending on application requirements.
- The CGC peripheral of the Synergy microcontrollers support Oscillator Stop Detection. If enabled in the application, the Oscillator Stop Detection function will automatically detect if the Main/PLL clock has stopped and switches operation to the MOCO. When enabling this functionality in the SSP a callback function has to be manually created by the user. The following steps detail this procedure.

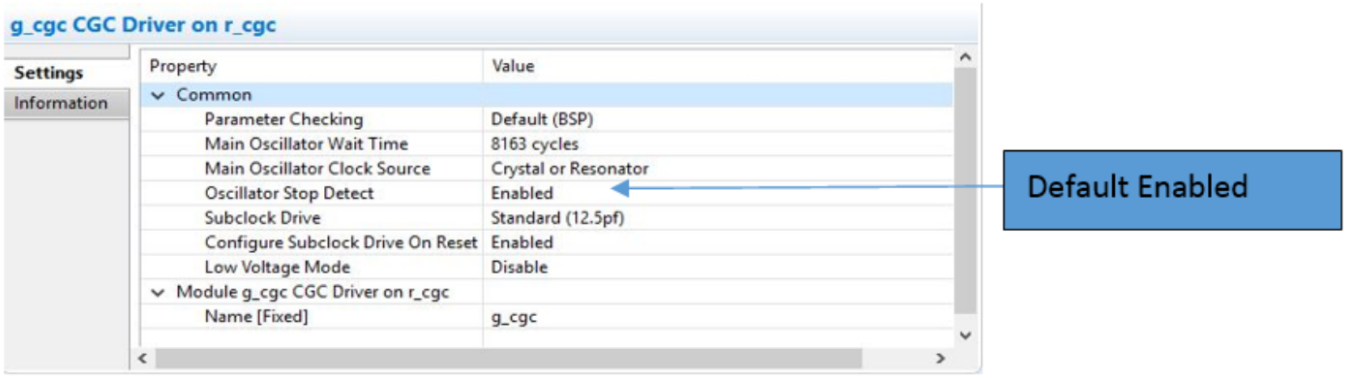


Figure 257: Oscillator Stop Detect Enable/Disable

Note

In SSP 1.4.0 and later, the Oscillator Stop Detect property is set to default as Enabled. Previous versions of SSP set the default to Disabled.

- If the Configure Subclock Drive on Reset is set to Disabled, then the subclock will not be configured at startup. In order to configure the subclock the user has to create a definition of a weakly linked function R_BSP_WarmStart(). In this function if the argument is "BSP_WARM_START_POST_C", the user can call the API cgc_api_t::clockStart clockStart to start the subclock.

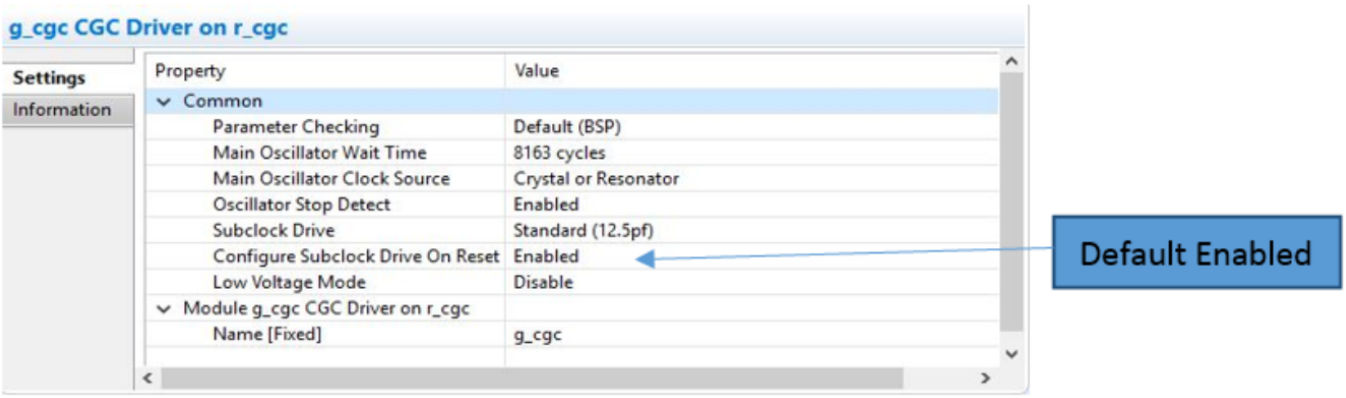


Figure 258: Configure Sub-Clock Drive on Reset

In the application code, create a callback function. In this example, it is called osc_stop_callback.

```
void osc_stop_callback(cgc_callback_args_t * p_args)
{
/* perform Oscillator Stop Detection processing */
}
```

Enable the oscillator stop detection by calling the API with the previously declared callback.

```
/* Enable the Osc Stop Detect functionality */
g_cgc.p_api->oscStopDetect( osc_stop_callback, true );
```

Enable the interrupt within the ICU

```
/* Osc Stop Detect is an NMI interrupt.  Enable the NMI in ICU */
R_ICU->NMIER_b.OSTEN = 1;
```

CGC HAL Module Limitations

Refer to the most recent SSP release notes for limitations on the use of this module.

4.2.9.4 Including the CGC HAL Module in an Application

This section describes how to include the CGC HAL module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

The CGC Driver is automatically added to the HAL/Common thread, so it only needs to be added to a new thread if it has been removed. (The default name for the CGC is g_cgc0. This name can be changed in the associated Properties window.)

CGC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_cgc CGC HAL on r_cgc	Threads	**New Stack> Driver> System> CGC Driver on r_cgc**

When the CGC Driver on r_cgc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are

individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

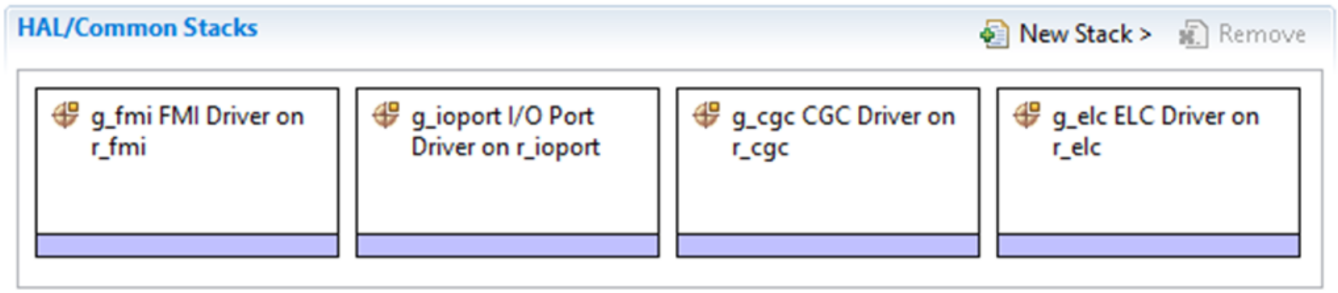


Figure 259: CGC HAL Module Stack

4.2.9.5 Configuring the CGC HAL Module

The CGC HAL module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CGC HAL Module on r_cgc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Main Oscillator Wait Time	3,35,67,131,259,547,1059,2147,4291,8163 cycles Default: 8163 cycles	Set to 0 if a resonator, or crystal, is used. Set to 1 if an external oscillator input is used.

Main Oscillator Clock Source	External Oscillator, Crystal or Resonator Default: Crystal or Resonator	Set to one of these values. It should be at least as long as the main clock stabilization time. This delay will be configured only if #define CGC_CFG_MAIN_OSC_CLOCK_SOURCE is set to 0, indicating that a resonator/ crystal is used. Set the main clock oscillation stabilization time to longer than or equal to the stabilization time recommended by the oscillator manufacturer.
Oscillator Stop Detect	Enabled, Disabled Default: Enabled	This allows the R_CGC_OscStopDetect function code to be generated if enabled. The user must call this function with a callback pointer to use this feature.
Subclock Drive	Middle (4.4pf), Standard (12.5pf) Default: Standard (12.5pf)	This setting is for matching the subclock oscillator drive capacitance based on the crystal parameters #define CGC_CFG_SUBCLOCK_ DRIVE.
Low Voltage Mode	Enable, Disable Default: Disable	Low voltage mode selection.
Name	g_cgc	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for the CGC HAL module can be desirable. For example, it might be useful to selectively turn clocks on or off or change frequency to optimize power and performance characteristics.

Note

Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

CGC HAL Module Clock Configuration

The default CGC HAL module clock frequencies that will be set by the BSP initialization process are configurable in the ISDE by using the **Clocks** tab in the configurator. Invalid selections are indicated in red when selected.

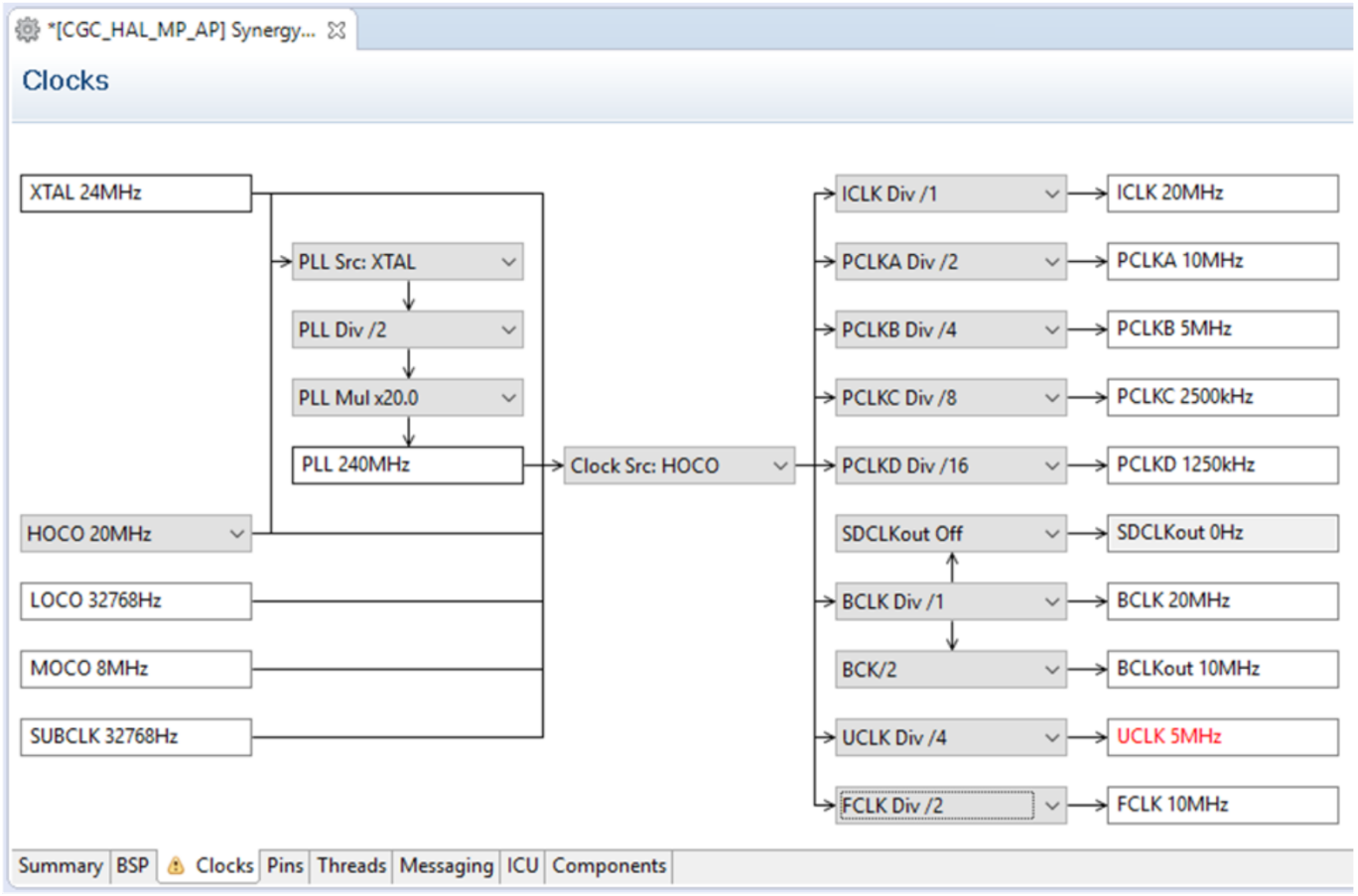


Figure 260: Default Clock Settings via the Clocks Tab

In this example, the Clock Source is HOCO, and various clock dividers are chosen for the peripheral clocks. If a valid USB Clock (UCLK) cannot be achieved, it is highlighted in RED. It should be noted that this is only advisory, and the project will still build, as such a clock frequency may be required.

CGC HAL Module Pin Configuration

The CGC peripheral module controls the output of BCLK and SDCLK signals. Use the **Clocks** tab to enable/disable this functionality. The BCLK_SDCLK I/O pin must be selected and configured as required via the **Pins** tab.

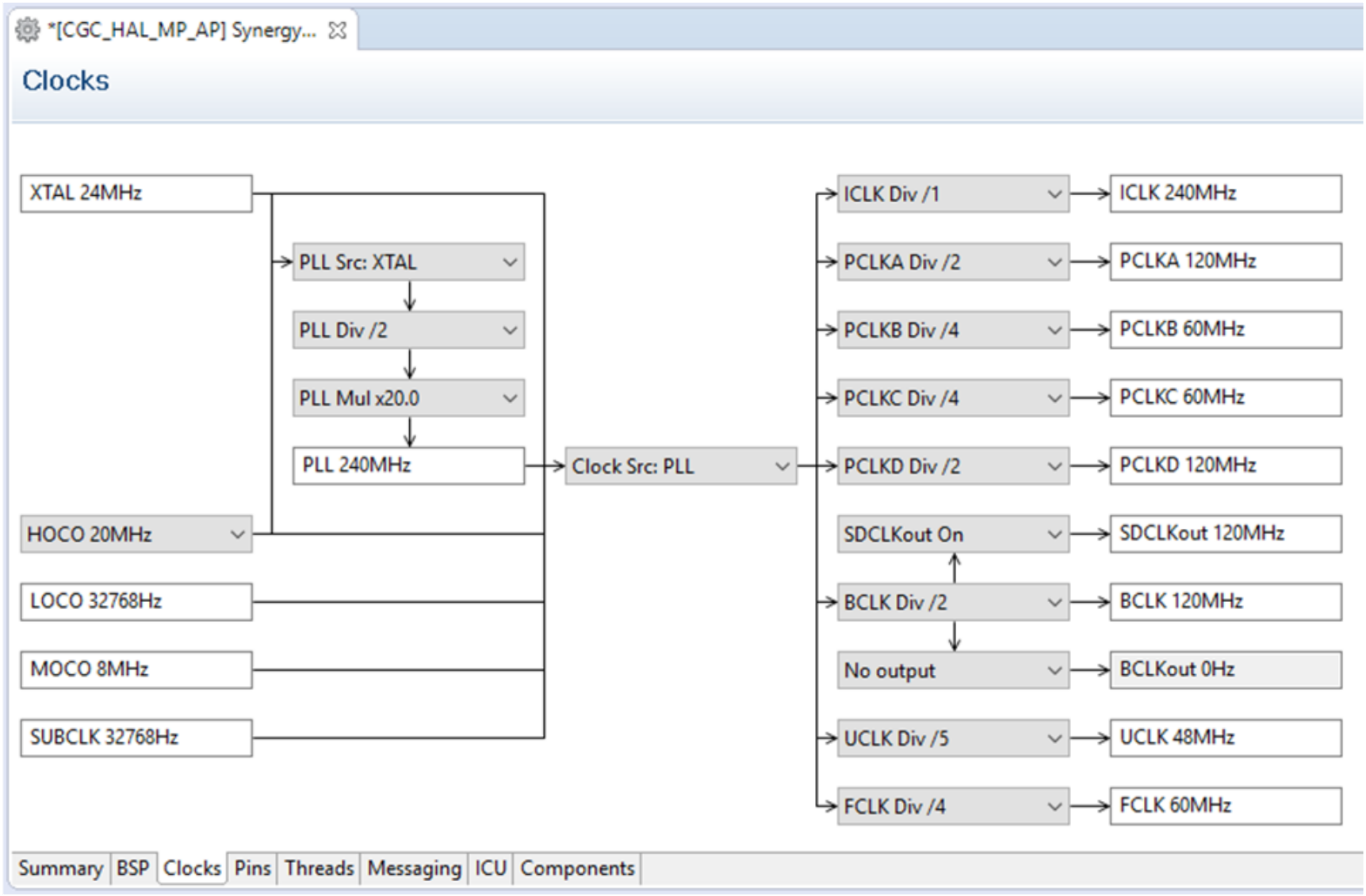


Figure 261: Enabling/Disabling SDCLK and BCLK via the Clock Tab

In this example, SDRAM Clock is enabled, BUS Clock is disabled.

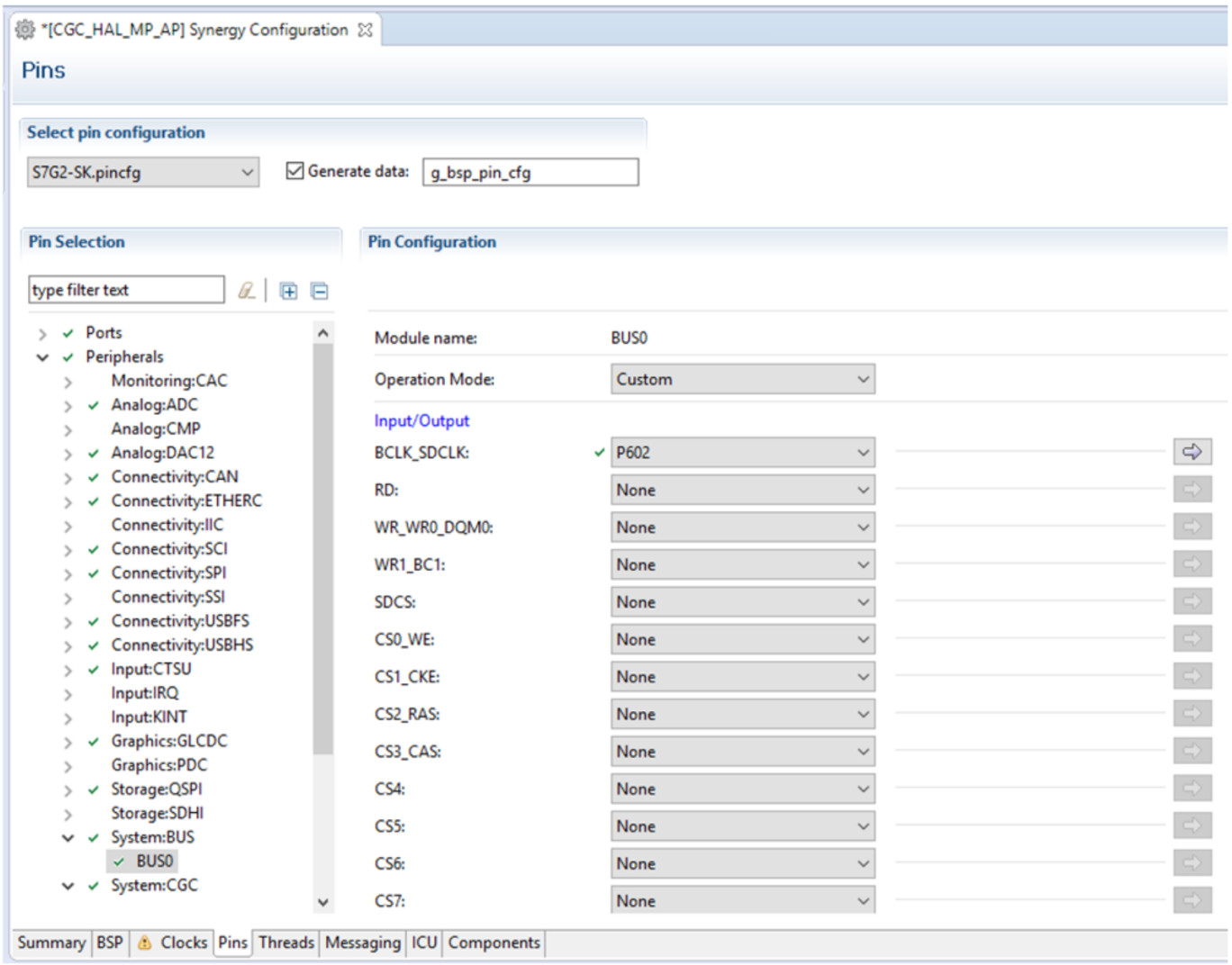


Figure 262: Enabling/Disabling SDCLK and BCLK via the Pins Tab

In this example, SDRAM Clock/BUS Clock is enabled on P602.

Additional pin settings associated with the CGC allow for the enabling/disabling of the external oscillator pins and the setting of the system Clock Out pin.

The Synergy devices can run from its on chip oscillators, in which case there is no requirement for the main clock external oscillator pins XTAL and EXTAL. These could be used as input pins by the application. The functionality of the sub clock external oscillator pins XCIN and XCOUT is fixed.

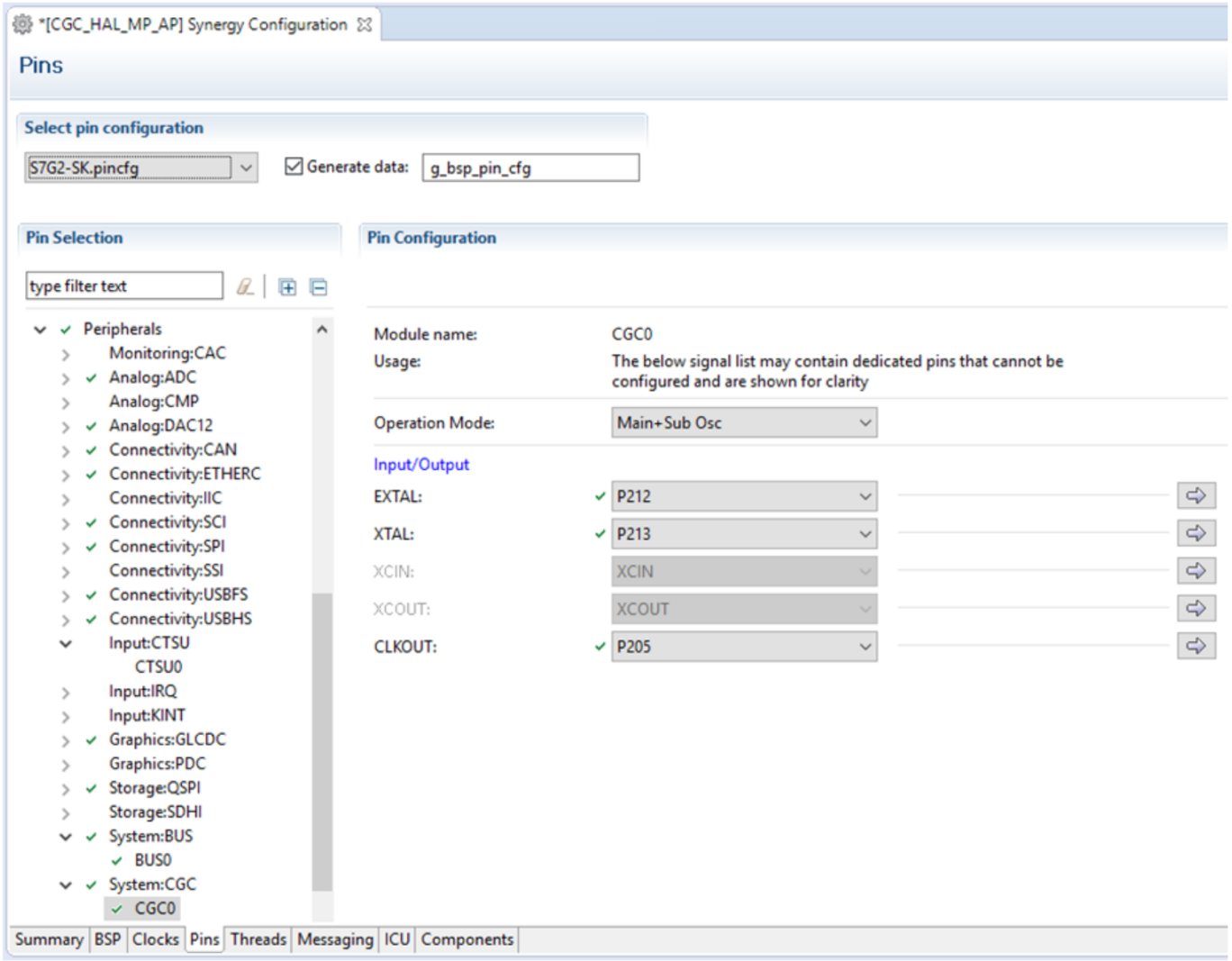


Figure 263: Enabling/Disabling EXTAL, XTAL and CLKOUT via the Pins Tab

In this example, an external Main Oscillator is used via pins P212 and P213 and the CLKOUT is enabled on P205.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and Synergy MCUs may have different available pin configuration settings.

4.2.9.6 Using the CGC Module in an Application

The typical steps in using the CGC HAL module in an application are:

1. The CGC is automatically set after the system reset.
2. Configure the clock as desired using the cgc_api_t::clocksCfg API.
3. Start clocks using the cgc_api_t::clockStart API if needed.
4. Stop clocks using the cgc_api_t::clockStop API if needed.
5. Other CGC functions as needed.

These common steps are illustrated in a typical operational flow diagram in the following figure:

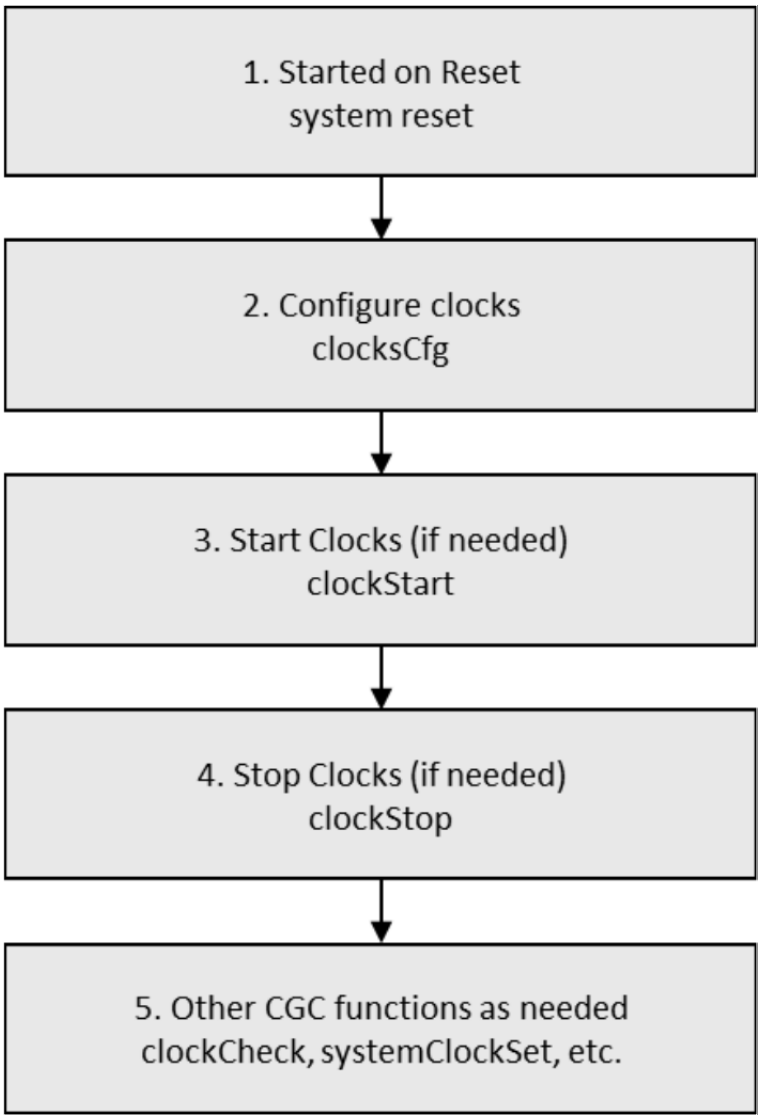


Figure 264: Flow Diagram of a Typical CGC HAL Module Application

4.2.10 CTSU v2 Driver

4.2.10.1 CTSU v2 HAL Module Introduction

The capacitive touch sensing unit version 2 HAL driver (r_ctsu v2) provides an API to control the CTSU peripheral. This module performs capacitance measurement based on various settings defined by the configuration. This module is configured via the QE for Capacitive Touch.

CTSU v2 HAL Module Features

- Supports both Self-capacitance multi scan mode and Mutual-capacitance full scan mode
- Scans may be started by software or an external trigger
- Returns measured capacitance data on scan completion

- Optional DTC support
- Built-in function that diagnoses its own circuit.

4.2.10.2 CTSU v2 HAL Module Configuration

Note

This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".

The following build time configurations are defined in ssp_cfg/r_ctsu_cfg.h:

Build Time Configurations for r_ctsu

Configuration	Options	Default	Description
Parameter Checking	- Default (BSP) - Enabled - Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support for using DTC	- Enabled - Disabled	Disabled	Enable DTC support for the CTSU module.
Interrupt priority level	MCU Specific Options		Priority level of all CTSU interrupt (CSTU_WR,CTSU_RD,CTSU_FN)

This module can be added to the Stacks tab via New Stack > Driver > CapTouch > CTSU Driver on r_ctsu.

Configurations for Driver > CapTouch > CTSU Driver on r_ctsu

Configuration	Options	Default	Description
Scan Start Trigger	MCU Specific Options		CTSU Scan Start Trigger Select

CTSU v2 HAL Module Interrupt Configuration

The first R_CTSU_Open function call sets CTSU peripheral interrupts. The user should provide a callback function to be invoked at the end of the CTSU scan sequence. The callback argument will contain information about the scan status.

CTSU v2 HAL Module Clock Configuration

The CTSU peripheral module uses PCLKB as its clock source. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Note

The CTSU Drive pulse will be calculated and set by the tooling depending on the selected transfer rate.

CTSU v2 HAL Module Pin Configuration

The TSn pins are sensor pins for the CTSU.

The TSCAP pin is used for an internal low-pass filter and must be connected to an external decoupling capacitor.

4.2.10.3 CTSU v2 HAL Module Usage Notes

CTSU v2 HAL Module Sensor ICO Correction

In order to improve the measurement accuracy, the correction coefficient is generated at the first R_CTSU_Open.

Therefore, the first R_CTSU_Open process takes about 40ms.

CTSU v2 HAL Module Initial Offset Tuning

CTSU v2 has a current offset mechanism to cancel the parasitic capacitance. This module automatically adjusts to be within the dynamic range of the Sensor ICO, taking into account the amount of current that changes with touch. This adjustment uses normal measurement process and requires several R_CTSU_ScanStart and R_CTSU_DataGet. R_CTSU_DataGet returns SSP_ERR_CTSU_INCOMPLETE_TUNING if it is being adjusted. The member "so" of ctsu_element_cfg_t is used as the starting point for adjustment, so if this value is appropriate, it can be completed quickly. Normally, this value uses the value adjusted by QE for Capacitive Touch.

CTSU v2 HAL Module Scan Trigger

Scanning of sensors may begin by either a software trigger or an external event initiated by the Event Link Controller (ELC). This trigger can be set with the member "cap" of ctsu_cfg_t. Typically, a software trigger is used. Common usage is to have a periodic timer initiate scans. For software triggers, a periodic timer such as the CMT is configured whose interval is large enough to allow for all sensors to be scanned and data to be updated. Software triggers are issued by calling R_CTSU_ScanStart. Using an external trigger is processed almost identically to using software triggers. Call R_CTSU_ScanStart before starting the timer to set the measurement and prepare for external trigger measurement. After that, when the timer is started, the measurement start trigger is applied.

CTSU v2 HAL Module Self-Capacitance Multi-Scan Mode

In self-capacitance mode each TS pin is assigned to one touch button. Electrodes of multiple TS pins can be physically aligned to create slider or wheel interfaces.

- Scan Order
  - The hardware scans the specified pins in ascending order.
  - For example, if pins TS05, TS08, TS02, TS03, and TS06 are specified in your application, the hardware will scan them in the order TS02, TS03, TS05, TS06, TS08.
- Element
  - -n element refers to the index of a pin within the scan order. Using the previous example, TS05 is element 2.
- Scan Time
  - Scanning is handled directly by the CTSU peripheral and does not utilize any main processor time.
  - It takes approximately 500us to scan a single sensor.
  - If DTC is not used additional overhead is required for the main processor to transfer data to/from registers when each sensor is scanned.

Set CTSU_MODE_SELF_MULTI_SCAN to "md" of ctsu_cfg_t. Also, add the number of terminals used for

this measurement to CTSU_CFG_NUM_SELF_ELEMENTS. For details, refer to the configuration and sample application output by QE for Capacitive Touch.

CTSU v2 HAL Module Mutual-Capacitance Full Scan Mode

In mutual-capacitance mode each TS pin acts as either a 'row' or 'column' in an array of sensors. As a result, this mode uses fewer pins when more than five sensors are configured. Mutual-capacitance mode is ideal for applications where many touch sensors are required, like keypads, button matrices and pads.

As an example, consider a standard phone keypad comprised of a matrix of four rows and three columns.

In mutual capacitance mode only 7 pins are necessary to scan 12 buttons. In self mode, 12 pins would be required.

- Scan Order
  - The hardware scans the matrix by iterating over the TX pins first and the RX pins second.
  - For example, if pins TS10, TS11, and TS03 are specified as RX sensors and pins TS02, TS07, and TS04 are specified as TX sensors, the hardware will scan them in the following sensor-pair order:
    TS03-TS02, TS03-TS04, TS03-TS07, TS10-TS02, TS10-TS04, TS10-TS07, TS11-TS02, TS11-TS04, TS11-TS07
- Element
  - An element refers to the index of a sensor-pair within the scan order. Using the previous example, TS10-TS07 is element 5.
- Scan Time
  - Because mutual-capacitance scans two patterns for one element it takes twice as long as self-capacitance (1ms vs 0.5ms per element).

Set CTSU_MODE_MUTUAL_FULL_SCAN to "md" of ctsu_cfg_t. Also, add the number of matrix used for this measurement to CTSU_CFG_NUM_MUTUAL_ELEMENTS. For details, refer to the configuration and sample application output by QE for Capacitive Touch.

CTSU v2 HAL Module Self Diagnosis

The CTSU peripheral has a built-in function that diagnoses its own inner circuit. The diagnostic requirements are providing 5 types of diagnosis. The diagnosis function is executed by calling the R_CTSU_Diagnosis() API function. This is executed independently from the other measurements and does not affect them.

Note

To enable the diagnosis function, set CTSU_CFG_DIAG_SUPPORT_ENABLE to 1.
A 27pF condenser should be connected externally.

4.2.10.4 CTSU v2 HAL Module Examples

CTSU v2 HAL Module Basic Example

This is a basic example of minimal use of the CTSU in an application.

```
volatile bool g_scan_flag = false;

void ctsu_callback (ctsu_callback_args_t * p_args)
```

```
{
  if (CTSU_EVENT_SCAN_COMPLETE == p_args->event)
  {
  g_scan_flag = true;
  }
}
void ctsu_basic_example (void)
{
  ssp_err_t err = SSP_SUCCESS;
  uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];

  err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);

  /* Handle any errors. This function should be defined by the user. */
  handle_error(err);



  while (true)
  {
  err = R_CTSU_ScanStart(&g_ctsu_ctrl);
  handle_error(err);
  while (!g_scan_flag)
  {
  /* Wait for scan end callback */
  }
  g_scan_flag = false;
  err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
  }
```

```
}
```

CTSU v2 HAL Module Multi-Configuration Example

This is a optional example of using both Self-capacitance and Mutual-capacitance configurations in the same project.

```
void ctsu_optional_example (void)
{
  ssp_err_t err = SSP_SUCCESS;
  uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS * 2)];
  err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
  handle_error(err);
  err = R_CTSU_Open(&g_ctsu_ctrl_mutual, &g_ctsu_cfg_mutual);
  handle_error(err);
  while (true)
  {
  R_CTSU_ScanStart(&g_ctsu_ctrl);
  while (!g_scan_flag)
  {
  /* Wait for scan end callback */
  }
  g_scan_flag = false;
  R_CTSU_ScanStart(&g_ctsu_ctrl_mutual);
  while (!g_scan_flag)
  {
  /* Wait for scan end callback */
  }
  g_scan_flag = false;
  err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
  handle_error(err);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
```

```
  err = R_CTSU_DataGet(&g_ctsu_ctrl_mutual, data);
  handle_error(err);
  if (SSP_SUCCESS == err)
  {
  /* Application specific data processing. */
  }
  }
}
```

CTSU v2 HAL Module Self Diagnosis Example

This is an example code of using self-diagnosis functionality.

```
ssp_err_t err;
uint16_t dummy;
/* Open the Diagnosis function */
R_CTSU_Open(g_qe_ctsu_instance_diagnosis.p_ctrl, g_qe_ctsu_instance_diagnosis.p_cfg);
/* Scan the Diagnosis function */
R_CTSU_ScanStart(g_qe_ctsu_instance_diagnosis.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;
err = R_CTSU_DataGet(g_qe_ctsu_instance_diagnosis.p_ctrl,&dummy);
/* Call diagnosis function when the return value of R_CTSU_DataGet is SSP_SUCCESS. */
if (SSP_SUCCESS == err)
{
  err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
  if ( SSP_SUCCESS == err )
  {
   /* Diagnosis is done successfully */
  }
}
```

4.2.11 CRC Driver

4.2.11.1 CRC HAL Module Introduction

The CRC HAL module provides a high-level API to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using industry standard polynomials.

CRC HAL Module Features

- CRC HAL module can calculate CRC on a block of data in memory.
- CRC HAL module can calculate CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode).
- CRC HAL module supports the following 8-and 16-bit CRC polynomials which operates on 8-bit data in parallel
  - $X^8 + X^2 + X + 1$ (CRC-8)
  - $X^{16} + X^{15} + X^2 + 1$ (CRC-16)
  - $X^{16} + X^{12} + X^5 + 1$ (CRC-CCITT)
- CRC HAL module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
  - $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ (CRC-32)
  - $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$ (CRC-32C)
- CRC HAL module can calculate CRC with LSB first or MSB first bit order.

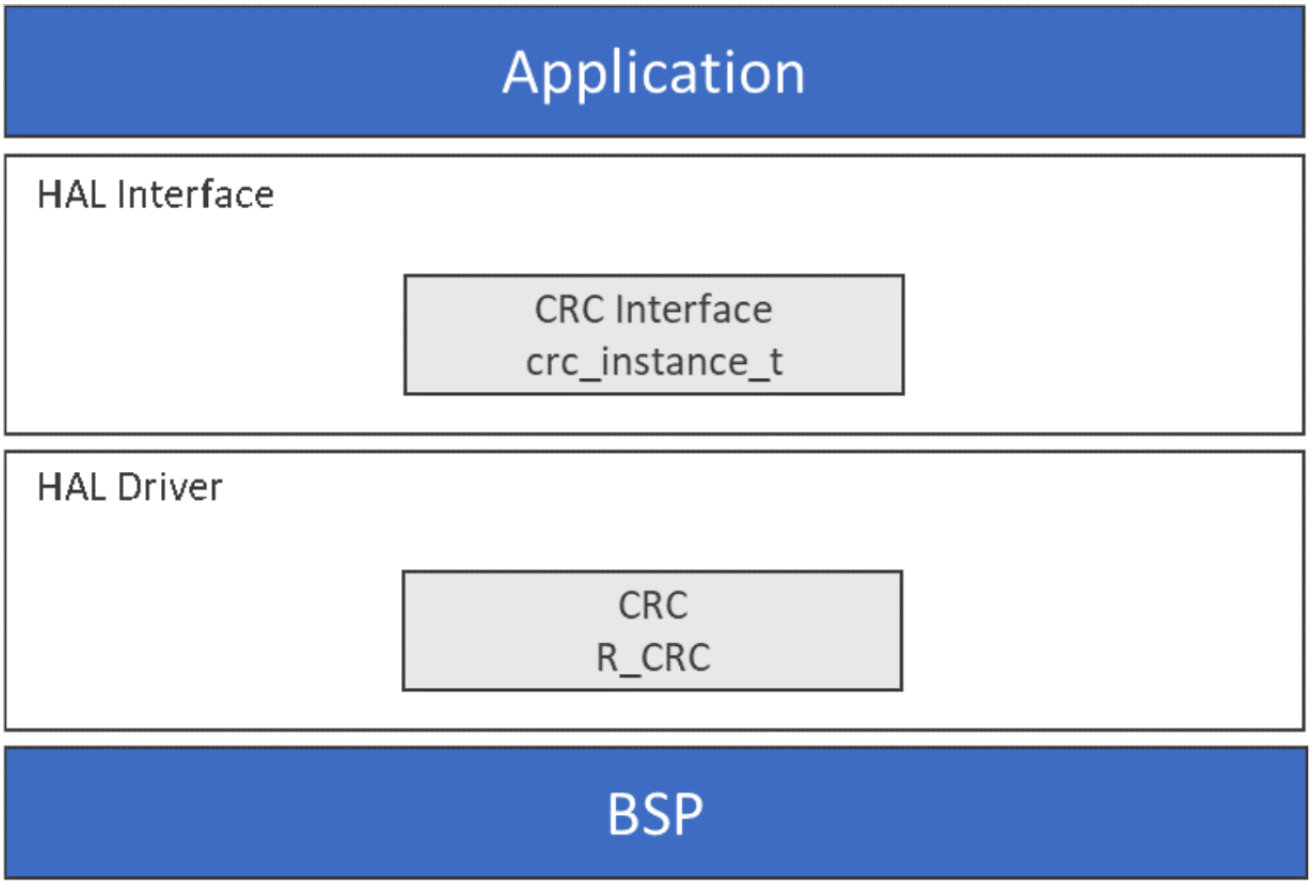


Figure 265: CRC HAL Module Block Diagram

CRC Hardware support details

The following hardware features are, or are not, supported by SSP for the CRC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	8-bit Data Size	32-bit Data Size	CRC Calculation Switching	Module Stop Function	CRC Snoop
S124	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

4.2.11.2 CRC HAL Module APIs Overview

The CRC HAL module defines APIs for opening, closing, enabling and calculating. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

CRC HAL Module API Summary

Function Name	Example API Call and Description
open	g_crc.p_api->open(g_crc.p_ctrl, g_crc.p_cfg); Open the CRC driver module.
close	g_crc.p_api->close(g_crc.p_ctrl); Close the CRC module driver.

crcResultGet	g_crc.p_api->crcResultGet(g_crc.p_ctrl, &result); Return the current calculated value.
snoopEnable	g_crc.p_api->snoopEnable(g_crc.p_ctrl, seed); Enable snooping.
snoopDisable	g_crc.p_api->snoopDisable(g_crc.p_ctrl); Disable snooping.
snoopCfg	g_crc.p_api->snoopCfg(g_crc.p_ctrl, g_crc.p_cfg); Configure the snoop channel and direction.
calculate	g_crc.p_api->calculate(g_crc.p_ctrl, &input_buffer, num_bytes, crc_seed, &crc_result); Perform a CRC calculation on a block of data.
versionGet	g_crc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Invalid argument error.
SSP_ERR_NOT_OPEN	The driver is not opened.
SSP_ERR_IN_USE	If driver is already open.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.11.3 CRC HAL Module Operational Overview

When the CRC HAL module is used to calculate the CRC value for a block of data in memory, the crc_api_t::calculate API can be used to take the input buffer pointer, length and the CRC seed value as input and outputs the calculated CRC value.

When the CRC HAL module is used to calculate the CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode), then first the module should be configured to be in snoop mode by calling the crc_api_t::snoopCfg followed by the crc_api_t::snoopEnable APIs. After the requested number of data is transmitted or received on the SCI channel, the calculated CRC value can be polled from the module using crc_api_t::crcResultGet API.

CRC HAL Module Important Operational Notes and Limitations

CRC HAL Module Operational Notes

General CRC HAL Operational Notes

- The CRC block does not use any interrupts.
- There is no clock configuration for the CRC module.
- There are no callbacks for the CRC module.
- When using 32 bit CRC polynomials for calculating CRC values of data block in memory, the data block is interpreted using little-endian byte order.

CRC HAL Snoop Mode Operational Notes

- The CRC snoop function monitors reads from (receive) and writes to (transfer) a specified I/O register address.
- It performs a CRC calculation on the serial data read from (receive) and written to (transfer) the register address automatically.
- The CRC calculation is performed 1 byte at a time. When the target I/O register address is accessed in words (16 bits) or long words (32 bits), the CRC code is generated on the lower byte (1 byte) of data.
- The CRC snoop mode is useful in monitoring writes to the serial transmit buffer, and reads from the serial receive buffer. If the user is trying to write data through the SCI interface, then the data will be present in the TDR register of that particular channel. When the user enables the Snoop mode with the proper configuration then the CRC module will automatically calculate the CRC value for the data present in that TDR register and then stores the generated CRC value in the CRC Data output register.

CRC HAL Module Limitations

- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.11.4 Including the CRC HAL Module in an Application

This section describes how to include the CRC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the CRC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the CRC Driver is g_crc0. This name can be changed in the associated Properties window.)

CRC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_crc0 CRC Driver on r_crc	Threads	**New Stack**> **Driver**> **Monitoring**> **CRC Driver on r_crc**

When the CRC Driver on r_crc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional

configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

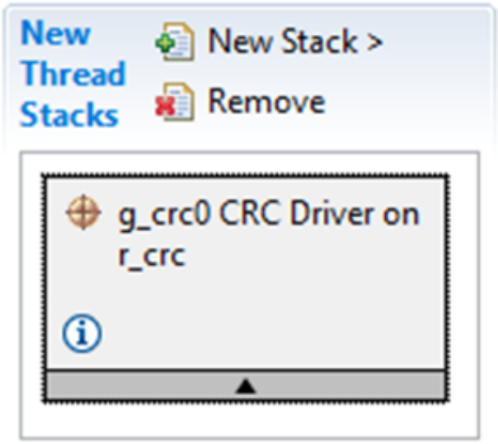


Figure 266: CRC HAL Module Stack

4.2.11.5 Configuring the CRC HAL Module

The CRC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CRC HAL Module on r_crc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g-crc0	Module name.

CRC Polynomial	CRC-8, CRC-16, CRC-CCITT, CRC-32, CRC-32C Default: CRC-32C	Specify the polynomial to use for calculation.
Bit Order	LSB, MSB Default: MSB	Specify the bit order of the calculation.
FIFO Mode	Enable, Disable Default : Disable	Enable this property when using SCI_UART with FIFO mode during CRC snoop operation.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.
CRC snoop mode operation for SCI_UART with FIFO must be enabled when the SCI channel supports FIFO mode.

CRC HAL Module Clock Configuration

The CRC HAL module is clocked via the Peripheral Clock A (PCLKA.)

The CRC HAL module does not support any APIs for setting the frequency at which it operates.

CRC HAL Module Pin Configuration

The CRC HAL module does not have any configurable pins.

4.2.11.6 Using the CRC HAL Module in an Application

There are two main types of CRC implementations- normal mode and snoop mode. The typical steps for each mode are shown below.

The typical steps in using the CRC HAL module in an application are:

1. Initialize the CRC HAL module using the crc_api_t::open API.
2. Compute the CRC HAL module using the crc_api_t::calculate API.
3. Close the CRC HAL module using the crc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

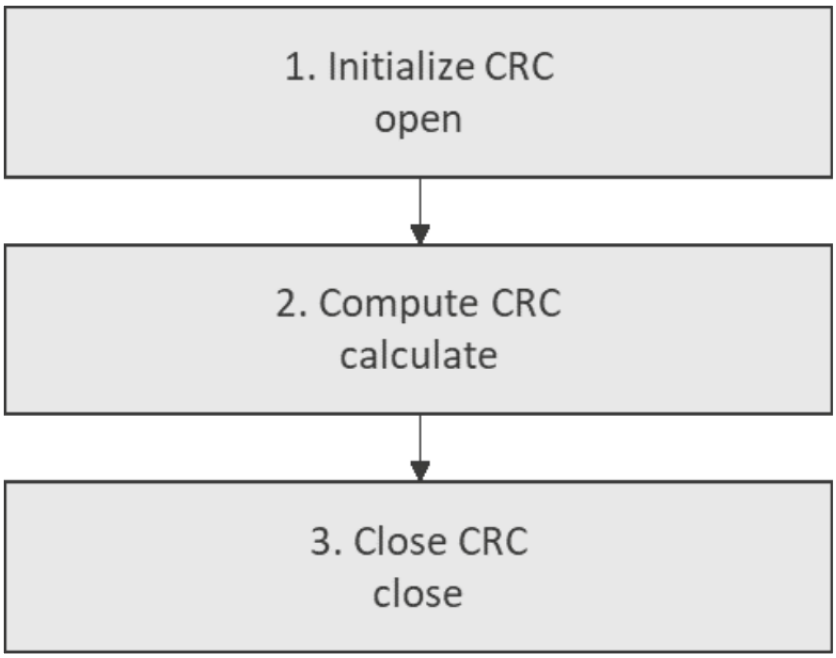


Figure 267: Flow Diagram of a Typical CRC HAL Module Application

The typical steps in using the CRC HAL module for computing in Snoop mode are:

1. Initialize the CRC HAL module using the crc_api_t::open API.
2. Snoop channel and snoop direction must be set manually by using snoop configuration structure crc_snoop_cfg_t.
3. Configure the CRC module to snoop an SCI channel (and its direction) using crc_api_t::snoopCfg API.
4. Enable snooping of the SCI channel using crc_api_t::snoopEnable API.
5. Once the required number of bytes are transmitted or received on the SCI channel, get the calculated CRC value using crc_api_t::crcResultGet API.
6. Disable the snooping operation using crc_api_t::snoopDisable API.
7. Close the CRC HAL module using the crc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

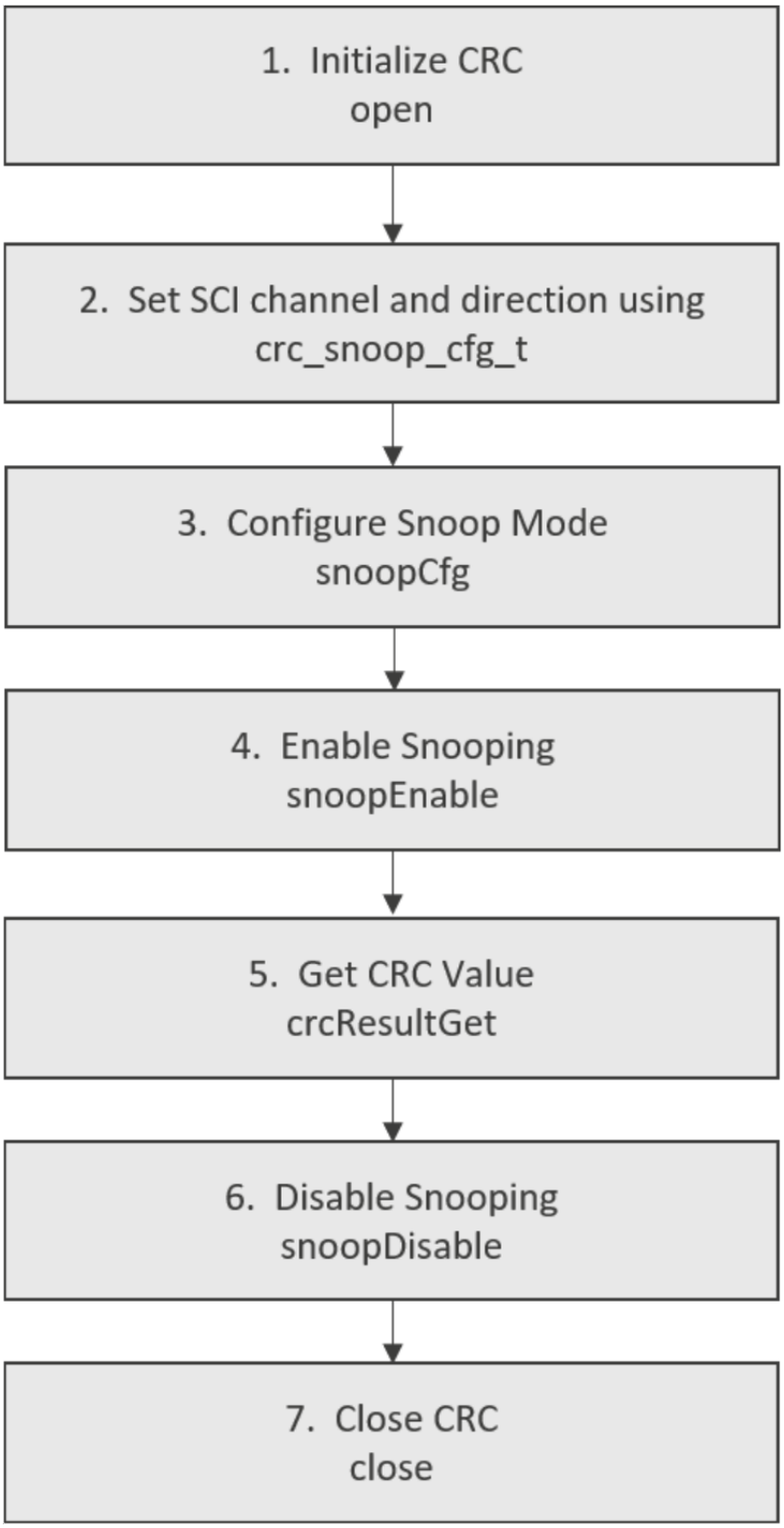


Figure 268: Flow Diagram of a Typical CRC HAL Module Application

4.2.12 DAC Driver

4.2.12.1 DAC HAL Module Introduction

The DAC HAL module provides a high-level API for digital-to-analog conversion applications and supports a dual-channel 12-bit D/A converter (DAC12) peripheral on Synergy MCUs.

DAC HAL Module Features

This module configures the dual-channel 12-bit D/A Converter (DAC12) to output one of 4096 voltage levels between the positive and negative reference voltages. The module includes configuration settings to:

- Set either a left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Enable or disable output amplifiers
- Enable or disable charge pump
- Operate in synchronous anti-interference mode with the Analog-to-Digital Converter (ADC) module.

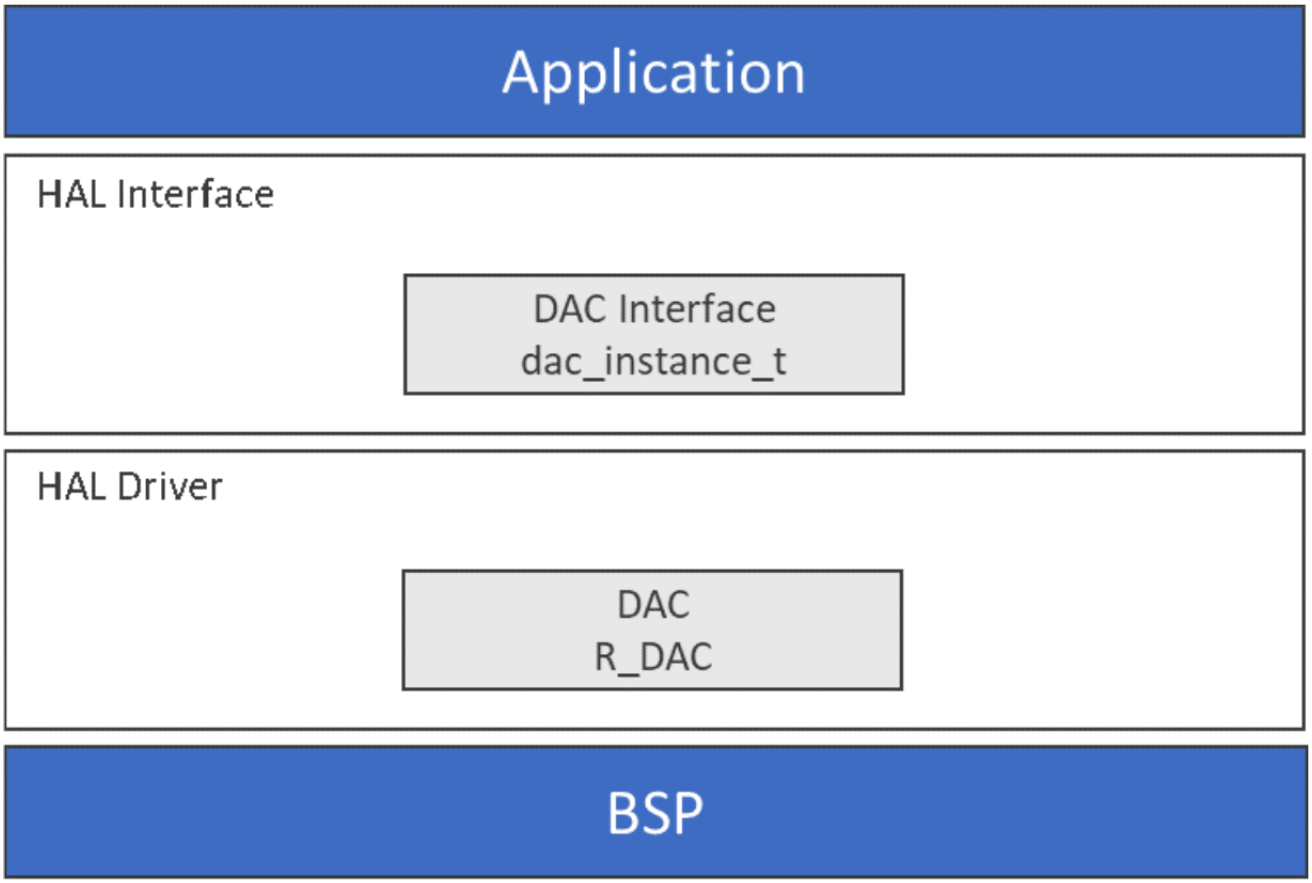


Figure 269: DAC HAL Module Block Diagram

DAC Hardware support details

The following hardware features are, or are not, supported by SSP for the DAC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	12-Bit	2 Channels Output	Module Stop Function	Event link function through ELC HAL driver *
S124	✓	N/A	✓	☒
S128	N/A	N/A	N/A	☒
S1JA	✓	N/A	✓	☒
S3A1	✓	N/A	✓	☒
S3A3	✓	N/A	✓	☒
S3A6	✓	N/A	✓	☒
S3A7	✓	✓	✓	☒
S5D3	✓	✓	✓	☒
S5D5	✓	✓	✓	☒
S5D9	✓	✓	✓	☒
S7G2	✓	✓	✓	☒

- Note: The ELC event could be used instead of calling the DAC start() interface. This would have to be programmed by the user by setting up the link rather than using the ELC API.

4.2.12.2 DAC HAL Module APIs Overview

The DAC HAL module defines APIs to open, close, start, stop and write to the DAC. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

DAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_dac.p_api->open(g_dac.p_ctrl, g_dac.p_cfg) Initial Configuration.
close	g_dac.p_api->close(g_dac.p_ctrl) Close the D/A Converter.
write	g_dac.p_api->write(g_dac.p_ctrl, val) Write Sample value to the D/A Converter.
start	g_dac.p_api->start(g_dac.p_ctrl) Start the D/A Converter if it has not been started yet.

stop	g_dac.p_api->stop(g_dac.p_ctrl) Stop the D/A Converter if the converter is running.
versionGet	g_dac.p_api->versionGet(&version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	Wrong parameter.
SSP_ERR_IN_USE	DAC resource is locked.
SSP_ERR_NOT_OPEN	The peripheral is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.12.3 DAC HAL Module Operational Overview

The DAC HAL module configures the dual-channel 12-bit D/A converter (DAC12) to output one of 4096 voltage levels between positive and negative reference voltages. The module can be used to configure the 12-bit output to left-or-right-justified format for 16-bit input data registers. The DAC HAL module can also enable or disable output amplifiers, or operate in synchronous anti-interference mode with the ADC HAL module.

DAC HAL Module Important Operational Notes and Limitations

DAC HAL Module Operational Notes

The DAC HAL module requires the following initialization steps:

- DAC module stop-bit cleared to zero.
- DAC channel output-enable set to one.

The DAC module stop-bit is cleared to zero at the time of an open call when the driver's instance counter is zero. The driver's instance counter is initialized to zero, incremented when a channel open returns successfully, and decremented when a channel close is called. The DAC module stop-bit is set to one when the driver's instance counter is decremented to zero on a close call.

The DAC channel output-enable is set to one when a channel write is called the first time after open was called successfully. The open call writes a zero to the dac_ctrl_t structure element channel_started. When write is called with channel_started cleared to zero, the DAC output-enable bit for that channel is set to one. The DAC output-enable for the channel is cleared to zero on close

and stop calls.

DAC HAL Module Limitations

- Pin configuration is not implemented for the DAC HAL module. Currently, DA0 and DA1 outputs are enabled by the reset values in the pin configuration control register's ASEL field.
- Voltage reference selection for the DAC HAL module is not implemented. Currently, no reference is selected by the reset values in the D/A VREF control register (DAVREFCR) which is a valid condition.
- Configuration of DAC input-event triggering for conversion is not currently implemented. The default reset value (zero) of the control register DAE bit allows individual triggering for each channel.
- Event signal input for synchronization of the DAC HAL module conversions is not currently implemented.
- The charge pump feature shall be disabled when DAC12 configured as ACMPHS or OPAMP input.
- Refer to the latest SSP Release Notes for operational limitations related to this module.

4.2.12.4 Including the DAC HAL Module in an Application

This section describes how to include the DAC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the DAC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the DAC Driver is g_dac0. This name can be changed in the associated Properties window.)

DAC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dac0 DAC Driver on r_dac	Threads	**New Stack> Driver> Analog> DAC Driver on r_dac**

When the DAC Driver on r_dac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

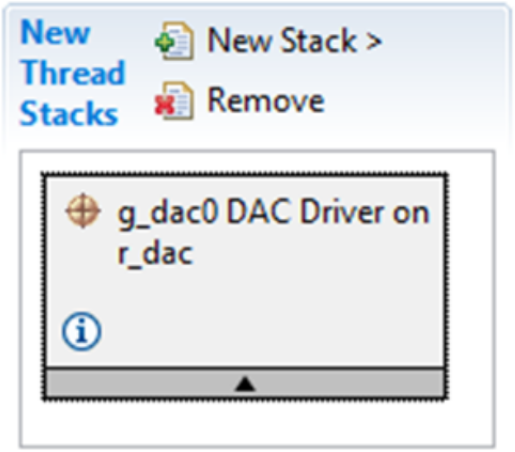


Figure 270: DAC HAL Module Stack

4.2.12.5 Configuring the DAC HAL Module

The DAC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DAC HAL Module on r_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac0	Module name.
Channel	0	Set to 0 for output DA0 or 1 for output DA1.

Synchronize with ADC	Enabled, Disabled Default: Disabled	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false false if power supply interference between the analog modules is not a problem, or if asynchronous conversion by the DAC Module is desired.
Data Format	Right Justified, Left Justified Default: Right Justified	Set to zero, if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.
Output Amplifier	Enable, Disable Default: Disable	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.
Charge Pump Enabled (Requires MOCO active)	Enable, Disable Default: Disable	Set to true, if charge pump hardware function is desired. Set to false to bypass charge pump hardware function.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DAC HAL Module Clock Configuration

The DAC HAL module does not require any specific clock configuration.

DAC HAL Module Pin Configuration

To use the DAC HAL module, the port pins for the channels receiving the analog input must be set as analog pins in the pin configurator. The following table lists a method to select pins within the SSP configuration window and the subsequent table illustrates configuration settings for DAC pins:

Pin Selection for the DAC HAL Module on r_dac

Resource	ISDE Tab	Pin selection Sequence
DAC	Pins	Select **Peripherals**> **Analog: DAC12**> **DAC120.**

Pin Configuration Settings for the DAC HAL Module on r_dac

Property	Value	Description
Module Name	DAC120	DAC Peripheral Module.

Operation Mode	Enabled, Disabled Default: Enabled	DAC Peripheral operation mode.
DA0	None, DA0 Default: None	DAC Output Pin.
DA1	None, DA1 Default: None	DAC Output Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.12.6 Using the DAC HAL Module in an Application

The typical steps in using the DAC HAL module in an application are:

1. Initialize the DAC HAL module using the dac_api_t::open API.
2. Write a data value using the dac_api_t::write API.
3. Start writing data using the dac_api_t::start API.
4. Continue writing data values as needed using the dac_api_t::write API.
5. Stop writing data using the dac_api_t::stop API.
6. Close the DAC HAL module using the dac_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

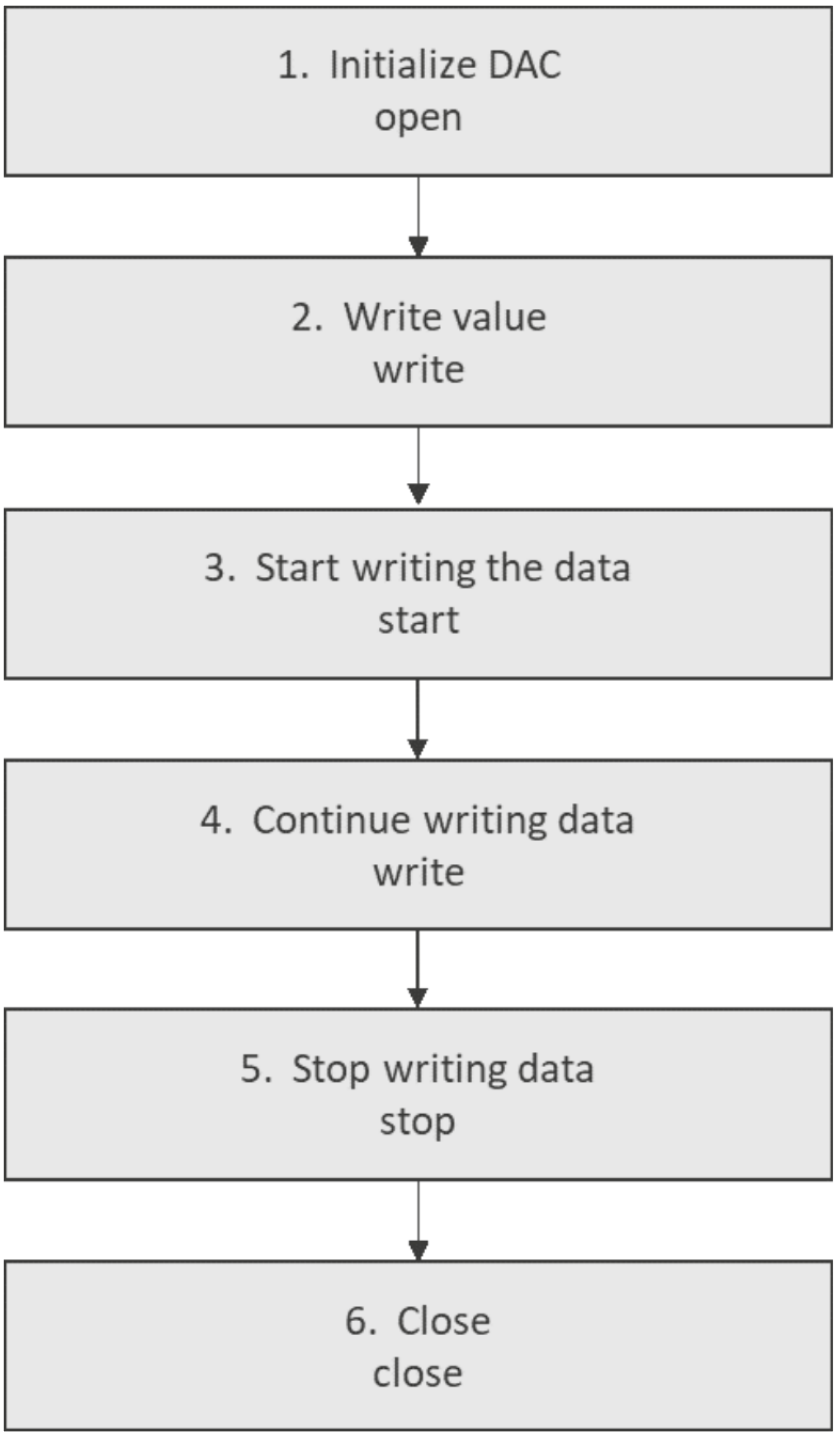


Figure 271: Flow Diagram of a Typical DAC HAL Module Application

4.2.13 DAC8 Driver

4.2.13.1 DAC8 HAL Module Introduction

The DAC8 HAL module provides a high-level API for digital-to-analog conversion applications and supports an 8-bit D/A converter (DAC8) peripheral on Synergy MCUs.

DAC8 HAL Module Features

- 8-Bit D/A Converter
- Left-Justified or Right-Justified Input Data Format
- Synchronization with the Analog-to-Digital Converter (ADC) module
- Multiple Operation Modes
    - Normal
    - Real-Time (Event Link)
    - Charge Pump Control

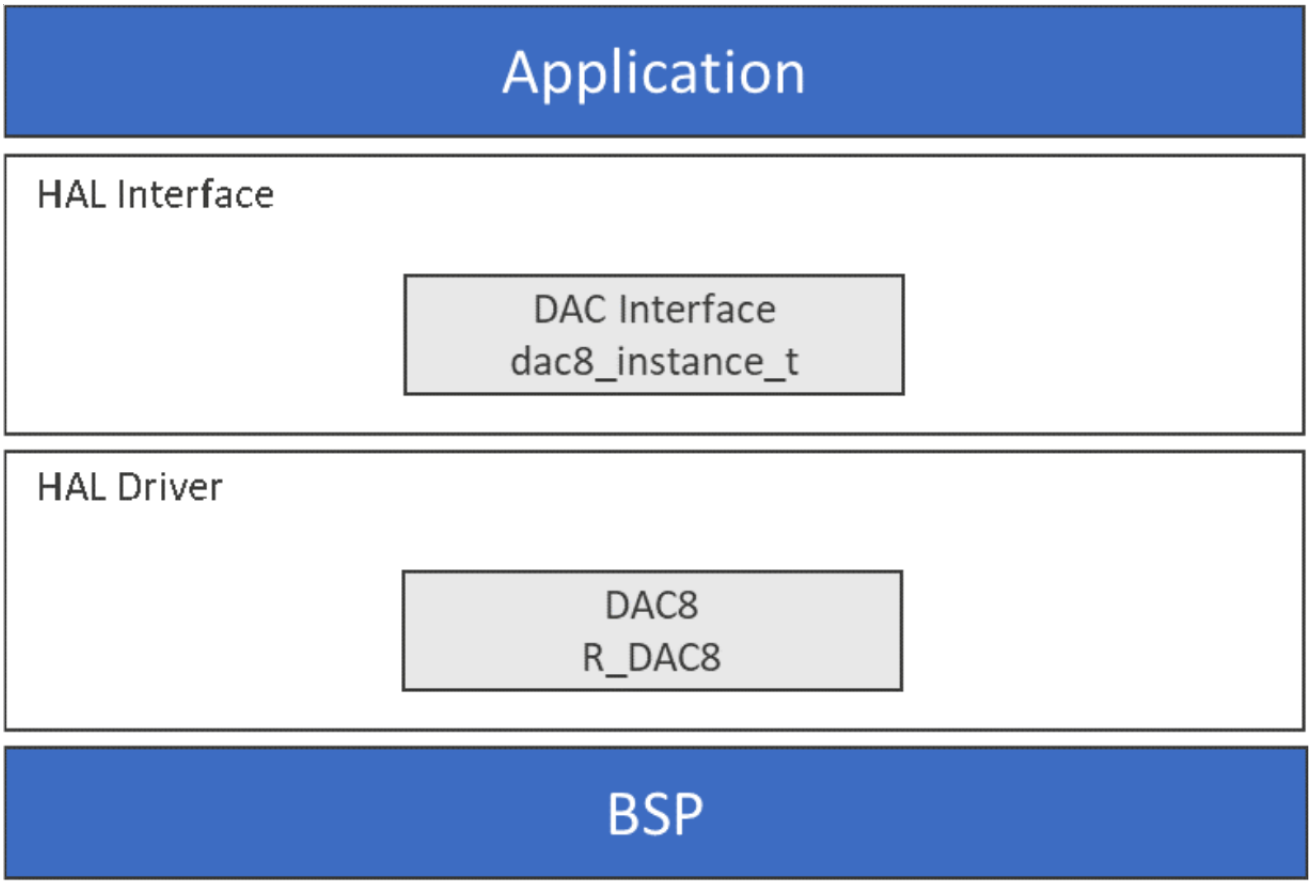


Figure 272: DAC8 HAL Module Block Diagram

DAC8 Hardware support details

The following hardware features are, or are not, supported by SSP for the DAC8.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A			Not supported by MCU		
MCU Group	8-Bit	2 Channels Output	3-Channel Output	Module-Stop Function	Event link function through ELC HAL driver *
S124	N/A	N/A	N/A	N/A	N/A
S128	✓	✓	✓	✓	✓
S1JA	✓	✓	N/A	✓	✓
S3A1	☒	☒	N/A	☒	☒
S3A3	✓	✓	N/A	✓	✓
S3A6	☒	☒	N/A	☒	☒
S3A7	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A	N/A

- Note: The ELC event could be used instead of calling the DAC start() interface. This would have to be programmed by the user by setting up the link rather than using the ELC API.

4.2.13.2 DAC8 HAL Module APIs Overview

The DAC8 HAL module defines APIs for opening, closing, starting, stopping and writing to the DAC. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

DAC8 HAL Module API Summary

Function Name	Example API Call and Description
open	g_dac8.p_api->open(g_dac8.p_ctrl, g_dac8.p_cfg Initial Configuration.
close	g_dac8.p_api->close(g_dac8.p_ctrl) Close the D/A Converter.
write	g_dac8.p_api->write(g_dac8.p_ctrl, val) Write Sample value to the D/A Converter.
start	g_dac8.p_api->start(g_dac8.p_ctrl) Start the D/A Converter if it has not been started yet.
stop	g_dac8.p_api->stop(g_dac8.p_ctrl) Stop the D/A Converter if the converter is running.

versionGet	g_dac8.p_api->versionGet(&version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_HW_LOCKED	DAC resource is locked.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	Wrong parameter.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Wrong channel selected.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.13.3 DAC8 HAL Module Operational Overview

The DAC8 HAL module configures the 8-bit D/A converter (DAC8) to output one of 256 voltage levels between positive and negative reference voltages. The driver can be configured to accept the 8-bit output data in left-or-right-justified format in a 16-bit input data. The driver supports two modes for the DAC.

- Normal mode – The D/A output is updated on writes to the data register.
- Real-Time (Event Link) – The D/A output is updated on an Event Link event. While in this mode the data register can be written at any time. An Event Link event triggers the start of conversion. Refer to the "ELC Interface" in the SSP User's Manual for more information.

To reduce the noise present in ADC readings the driver can configure synchronous anti-interference mode with the ADC module. This reduces conversion noise by disabling the DAC charge while the ADC is sampling. Check the hardware manual to determine if this feature is supported.

For operation at low AVCC voltage the driver can enable or disable the hardware charge pump.

Real-Time Mode

In real-time mode the output voltage is only changed on a signal from the ELC peripheral. Note when using real time mode the first call to dac_api_t::write() will set the initial output voltage.

As an illustration, the following code shows how to link the DAC8 output on channel 0 to the ELC software event and trigger the output to change.

```
/* Open DAC8 driver.  */
g_dac8_0.p_api->open(g_dac8_0.p_ctrl, &g_dac8_0.p_cfg);
```

```
/* Link DAC8 output 0 to software event 0.
   In a real application, this feature would be used to link DAC8 output
   to a hardware event. A software event is used here for simplicity. */
g_elc_on_elc.linkSet(ELC_PERIPHERAL_DA80, ELC_EVENT_ELC_SOFTWARE_EVENT_0);
/* Set initial output voltage, the output is immediately updated
   because this is the first write since open. */
g_dac8_0.p_api->write(g_dac8_0.p_ctrl, initial_dac_value);
while (true) {
/* Set the next output value, the output voltage is not updated until
       an ELC event occurs. */
g_dac8_0.p_api->write(g_dac8_0.p_ctrl, next_dac_value);
/* Generate software ELC event to update DAC output. */
g_elc_on_elc.softwareEventGenerate(ELC_SOFTWARE_EVENT_0);
R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
}
```

DAC8 HAL Module Important Operational Notes and Limitations

DAC8 HAL Module Operational Notes

- The DAC8 channel output is enabled during dac_api_t::start() and dac_api_t::write() and disabled during dac_api_t::stop() and dac_api_t::close().

DAC8 HAL Module Limitations

- The DAC8 driver does not configure the ELC peripheral for real-time mode. The user will need to configure the Event Link Controller in addition to enabling real-time mode in the DAC8 module.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.13.4 Including the DAC8 HAL Module in an Application

This section describes how to include the DAC8 HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the DAC8 Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the DAC8 Driver is g_dac8_0. This name can be changed in the associated Properties window.)

DAC8 HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dac8_0 DAC Driver on r_dac8	Threads	**New Stack> Driver> Analog> DAC Driver on r_dac8**

When the DAC8 Driver on r_dac8 is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

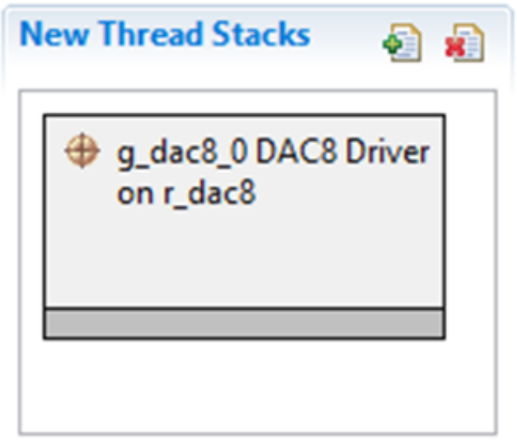


Figure 273: DAC8 HAL Module Stack

4.2.13.5 Configuring the DAC8 HAL Module

The DAC8 HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DAC8 HAL Module on r_dac8

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac8_0	Module name.
Channel	0	Channel selection.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Choose whether to sync with the ADC module.
Data Format	Right Justified, Left Justified Default: Right Justified	Data format selection.
DAC Mode	Normal Mode, Real-time (Event Link) Mode Default: Normal Mode	DAC mode selection.
Charge Pump Enabled (Requires MOCO active)	Enabled, Disabled Default: Enabled	Enable or disable the charge pump.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DAC8 HAL Module Clock Configuration

The DAC8 HAL module does not require a specific clock configuration.

DAC8 HAL Module Pin Configuration

To use the DAC8 HAL module, the port pins for the channels receiving the analog input must be set as analog pins in the pin configurator. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the DAC pins:

Pin Selection for the DAC8 HAL Module on r_dac8

Resource	ISDE Tab	Pin selection Sequence
DAC8	Pins	Select **Peripherals**> **Analog:DAC8**> **DAC80.**

Note

The selection sequence assumes DAC80 is the desired hardware target for the driver.

Pin Configuration Settings for the DAC8 HAL Module on r_dac8

Property	Value	Description
Module Name	DAC80	DAC Peripheral Module.

Operation Mode	Enabled, Disabled Default: Enabled	DAC Peripheral operation mode.
DA0	None, DA0 Default: None	DAC Output Pin.
DA1	None, DA1 Default: None	DAC Output Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.13.6 Using the DAC8 HAL Module in an Application

The typical steps in using the DAC8 HAL module in an application are:

1. Initialize the DAC8 HAL module using the dac_api_t::open API.
2. Write a value using the dac_api_t::write API.
3. Start a conversion using the dac_api_t::start API.
4. Continue writing values as needed using the dac_api_t::write API.
5. Stop conversion using the dac_api_t::stop API.
6. Use the dac_api_t::close call to power down the peripheral.

These common steps are illustrated in a typical operational flow diagram in the following figure:

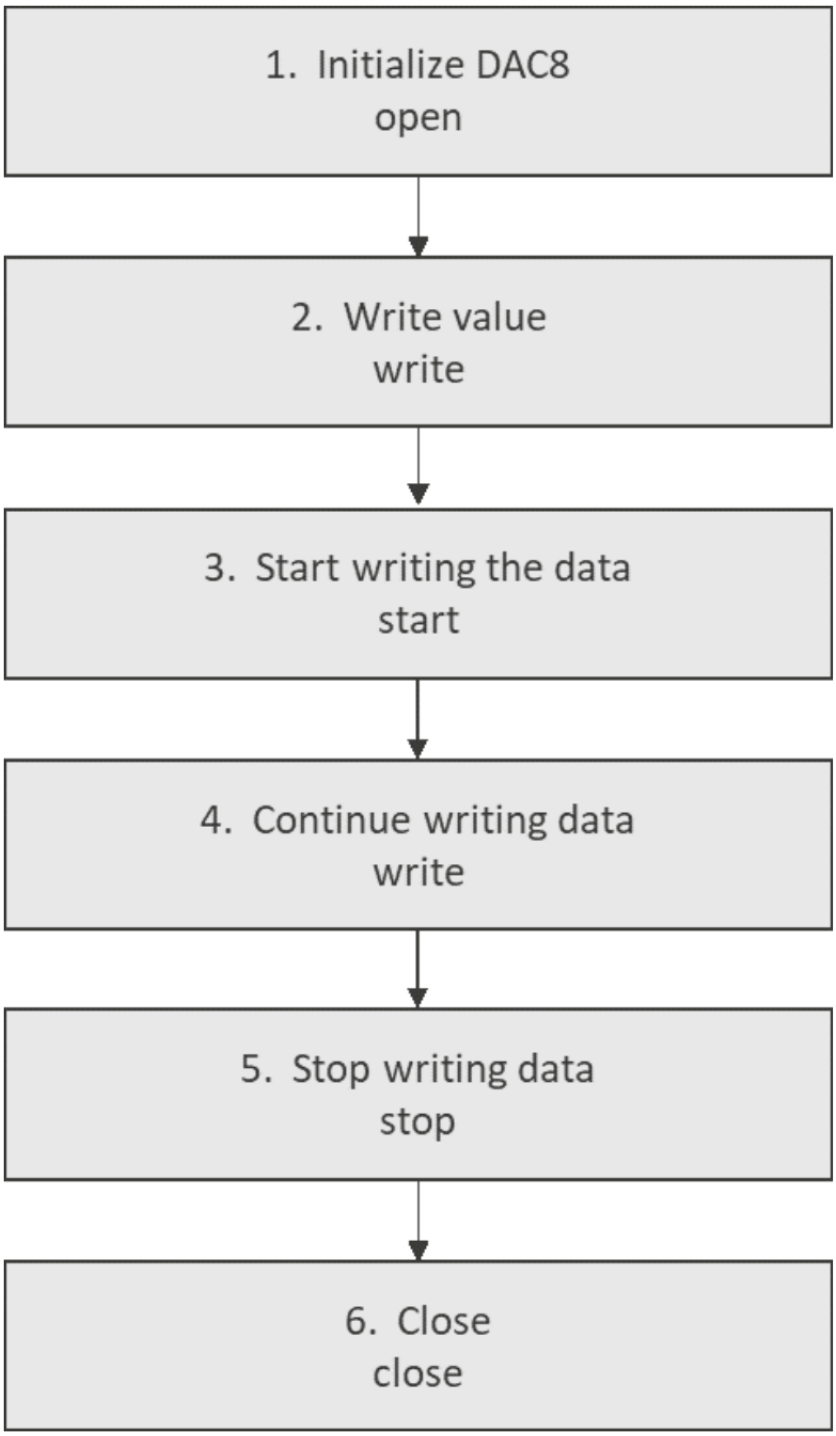


Figure 274: Flow Diagram of a Typical DAC8 HAL Module Application

4.2.14 Display Driver

4.2.14.1 GLCDC HAL Module Introduction

The Graphics LCD Controller (GLCDC) HAL module provides a high-level API for graphics display applications and uses the Graphics LCD Driver peripheral on the Synergy MCU. A user-defined callback can be created to handle frame buffer switching and underflow detection.

GLCDC HAL Module Features

The GLCDC HAL supports the following features:

- Supports LCD panels with RGB interface (up to 24 bits) and sync signals (HSYNC, VSYNC, and Data Enable (optional))
- Supports various color formats for input graphics planes (RGB888, ARGB888, RGB565, ARGB1555, ARGB4444, CLUT8, CLUT4, CLUT1)
- Supports the Color Look-Up Table (CLUT) usage for input graphics planes with 512 words (32 bits/word)
- Supports various color formats for output (RGB888, RGB666, RGB565, Serial RGB)
- Can input two graphics planes on top of the background plane and blend them on the screen
- Generates a dot clock to the panel. The clock source is selectable from internal or external (LCD_EXTCLK)
- Supports brightness adjustment, contrast adjustment, and gamma correction
- Supports GLCDC interrupts to handle frame buffer switching or underflow detection

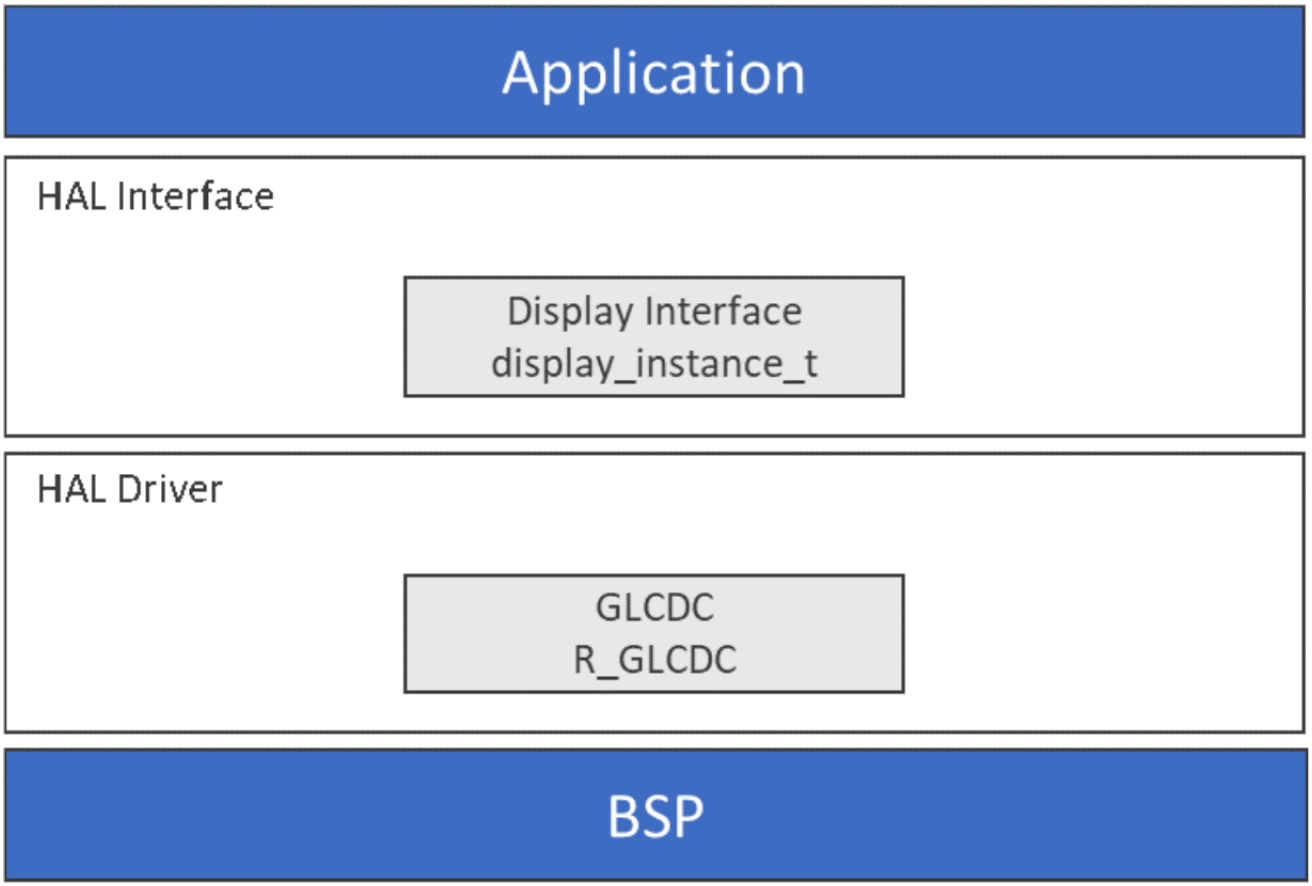


Figure 275: GLCDC HAL Module Block Diagram

GLCDC Hardware support details

The following hardware features are, or are not, supported by the SSP for the GLCDC:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Single Color Background Plane	Graphics 1 Plane	Graphics 2 Plane	Support 16 bit per pixel graphics	Support 32 bit per pixel graphics	Support 1-bit LUT
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Support 4-bit LUT	Support 8-bit LUT	Support All Pixel formats	Supports Alpha Blending	Video Signal Timing Adjustment	Supports All Output Data formats
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A

S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Supports All Dithering Modes	Support output of VSYNC, HSYNC **and Horizontal Data Enable**	Supports Brightness and Contrast	Supports Gamma Correction
S124	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A
S5D9	✓	✓	✓	✓
S7G2	✓	✓	✓	✓

4.2.14.2 GLCDC HAL Module APIs Overview

The GLCDC HAL module defines APIs for opening, closing, starting, stopping and controlling the display of information on an LCD panel. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

GLCDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_display.p_api->open (g_display.p_ctrl, g_display.p_cfg); Open display device.
close	g_display.p_api->close (g_display.p_ctrl); Close display device.

start	g_display.p_api->start(g_display.p_ctrl); Display start.
stop	g_display.p_api->stop(g_display.p_ctrl); Display stop.
layerChange	g_display.p_api->layerChange(g_display.p_ctrl, &layercng) Change layer parameters at runtime.
correction	g_display.p_api->correction(g_display.p_ctrl, &display_correction) Color correction.
clut	g_display.p_api->clut(g_display.p_ctrl, &clut) Set CLUT for display device.
statusGet	g_display.p_api->statusGet(g_display.p_ctrl, &status) Get status for display device.
versionGet	g_display.p_api->versionGet(&version) Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_HW_LOCKED	GLCDCC resource is locked.
SSP_ERR_CLOCK_GENERATION	Dot clock cannot be generated from clock source.
SSP_ERR_INVALID_TIMING_SETTING	Invalid panel timing parameter.
SSP_ERR_INVALID_LAYER_SETTING	Invalid layer setting found.
SSP_ERR_INVALID_LAYER_FORMAT	Invalid format is specified.
SSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction setting found.
SSP_ERR_NOT_OPEN	The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed when the GLCDC is updating register values internally.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_OPENED.

SSP_ERR_INVALID_CLUT_ACCESS	Illegal CLUT entry or size is specified.
p_version	The version number.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.14.3 GLCDC HAL Module Operational Overview

The GLCDC HAL module controls an LCD panel. The following figure shows an overview of the graphics data flow using the GLCDC HAL module. The module supports reading graphics frame image data from memory (up to two frames) and blending those images on top of the monochrome background screen. The driver supports CLUT memory and specifies the graphic frame format for the CLUT.

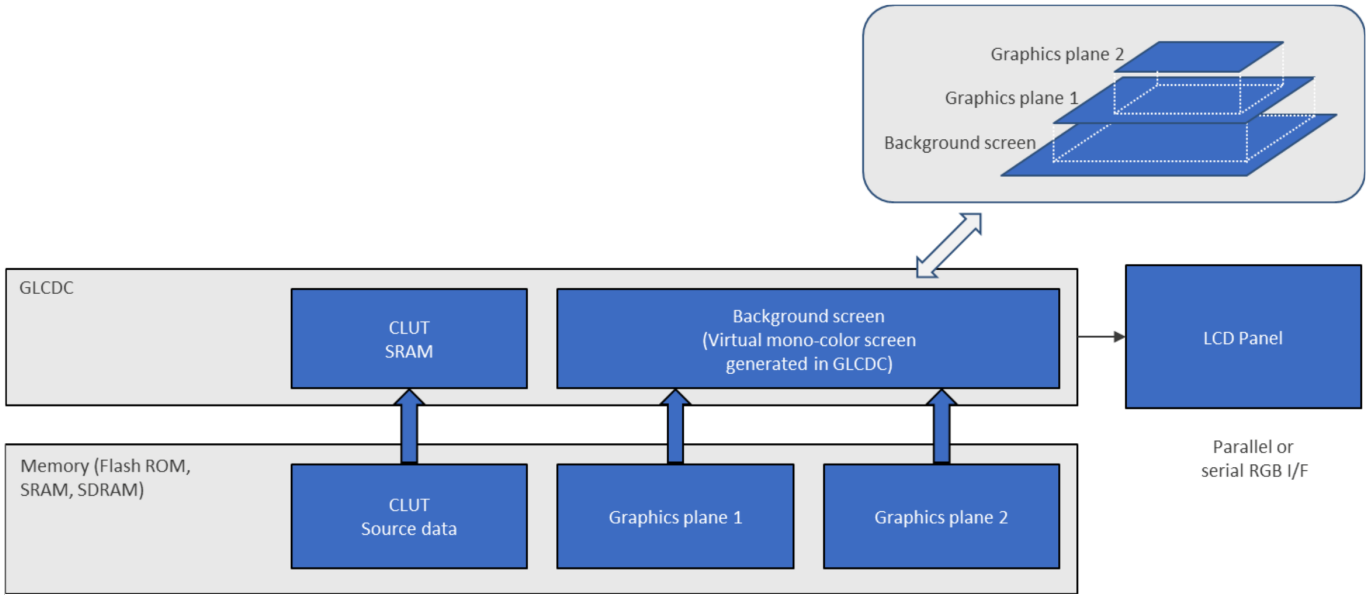


Figure 276: GLCDC HAL Module Data Flow

The following figure shows a display system with a ping-pong frame buffer. It is recommended to have more than two frame buffers in a display system to avoid image tearing, which happens in a single frame buffer display system. In such designs, the GLCDC hardware can read a graphics frame image from one of the frame buffers while the image drawing engines (for example, DRW and JPEG), CPU or DMAC/DTC transfer a graphics frame image to another frame buffer simultaneously. The module supports frame buffer toggling by the display_api_t::layerChange API at run-time.

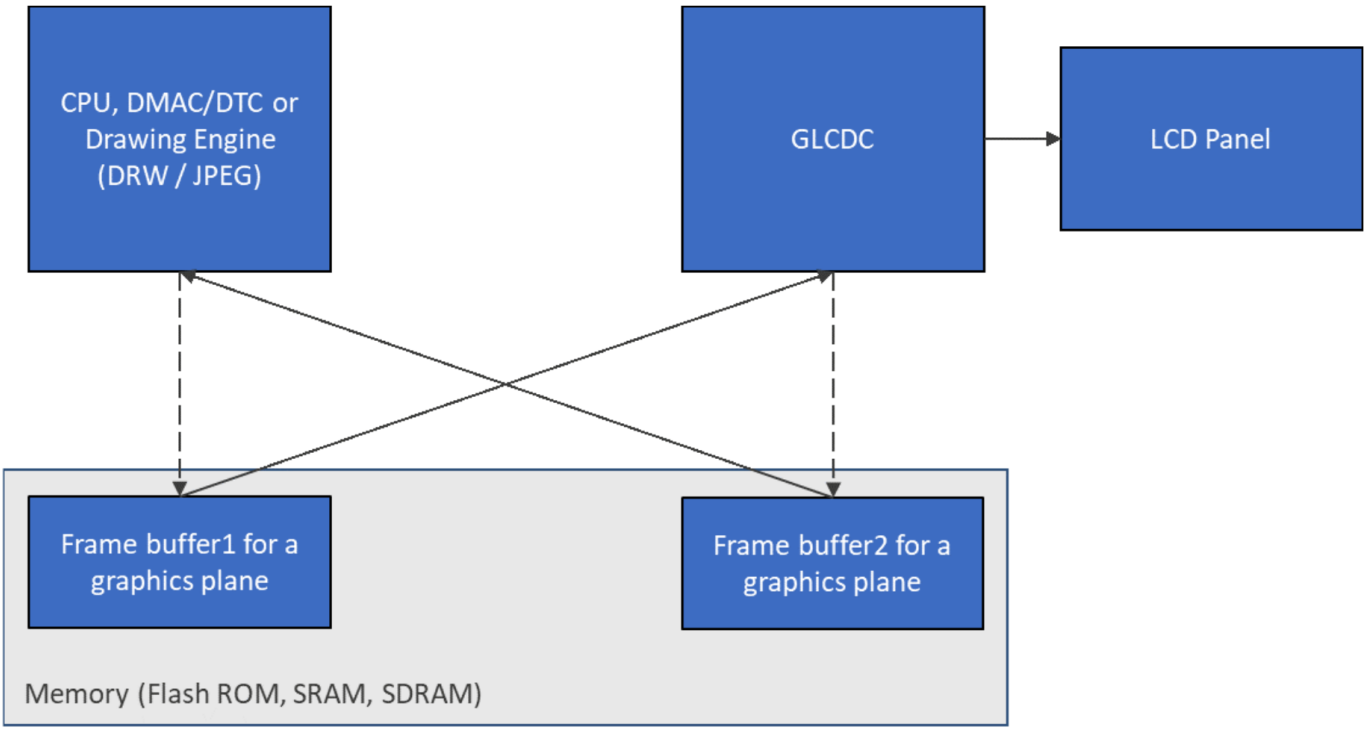


Figure 277: GLCDC HAL Module Ping-Pong Buffer System

Screen Format

The following figure shows the relationship between the LCD screen format and LCD timing parameters of the GLCDC module. The module has generic timing parameters for the LCD panel setting that support a variety of LCD panels.

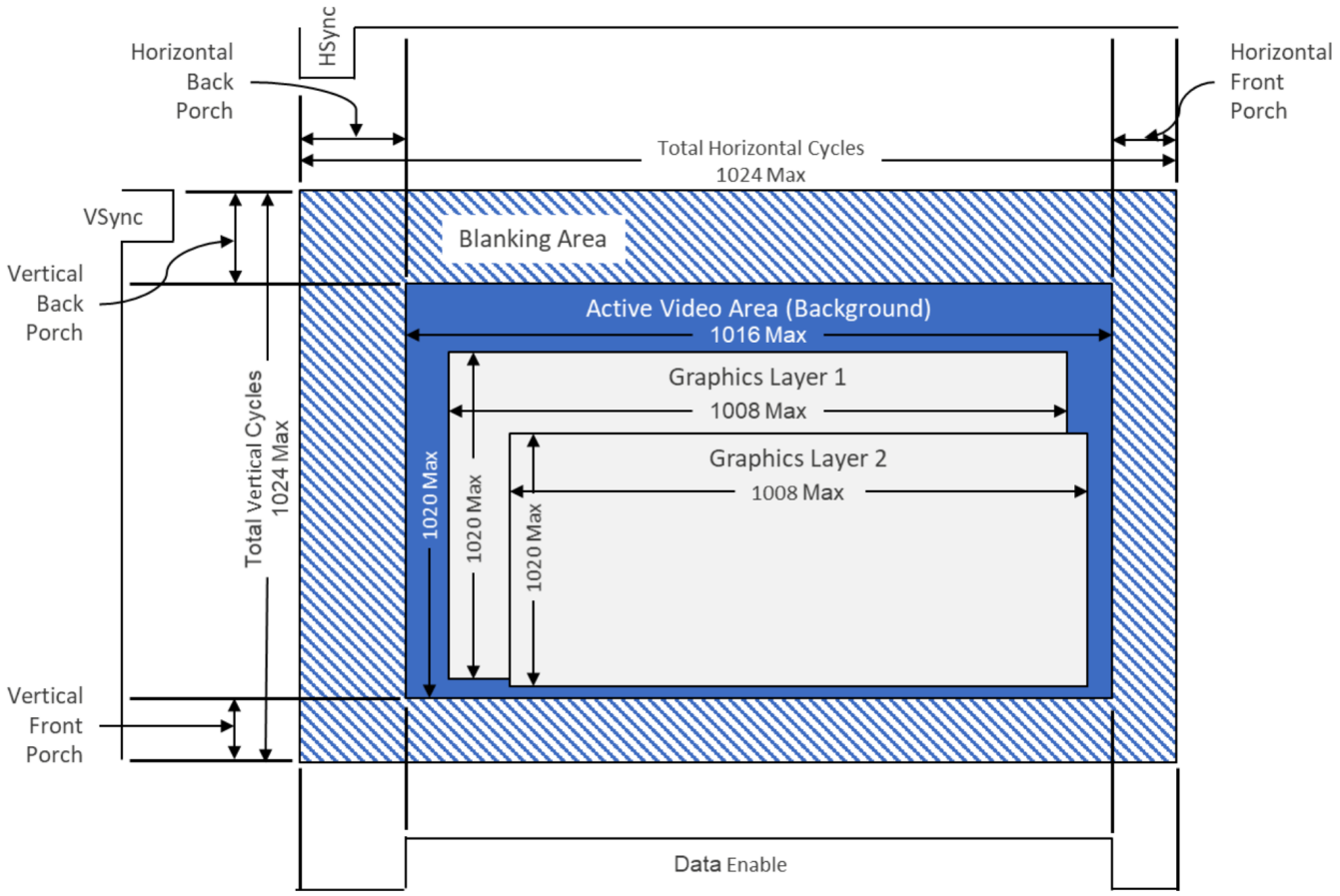


Figure 278: GLCDC HAL Module Screen Format

Front Porch Period

The GLCDC module does not have a setting for horizontal/vertical front porch cycles/lines. Those cycles/lines must be included in the total horizontal cycles/vertical lines setting.

Note

The module requires setting the back porch cycles/lines based on the GLCDC hardware specification. Since typical LCD panels have a greater number of back porch cycles/lines than described, this is not a true limitation of the module.

- Number of the horizontal back porch cycles >= 3
- Number of the vertical back porch lines >= 2

Example Parameter Settings

PE-HMI1 v2.0 board (LXD Research & Display, LLC, M7504A):

The following example adjusts the horizontal total cycles, vertical total lines, and panel clock division ratio to generate an LCD panel refresh rate of 60 Hz. Regarding symbols for the LCD panel, see the M7504A data sheet.

LCD Panel Parameter Settings - PE-HMI1 v2.0 Board

DK-S7G2 v3.0 board (LXD Research & Display, LLC, M7190A):

The following example adjusts the horizontal total cycles, vertical total lines and panel clock division ration to generate an LCD panel refresh rate of 60 Hz. Regarding symbols for the LCD panel, see the M7190A data sheet.

LCD Panel Parameter Settings - DK-S7G2 v3.0 Board

SK-S7G2 v2.0 board (ILI Technology Corp., IL9341C):

The following example sets the horizontal total cycles, and vertical total lines as large as allowed for the panel for an LCD panel refresh rate of about 76.8 Hz. Regarding symbols for the LCD panel, see the LIL9314V data sheet.

LCD Panel Parameter Settings - SK-S7G2 v2.0 Board

Note

The input horizontal size and stride are intentionally set to 256 pixels, even though the parameter should be 240 pixels for the panel. This is because a horizontal line has to be 64-byte aligned for GLCDC hardware. Only 240 pixels from the beginning in a line are valid and rest of pixels in the line (16 pixels) are don't care.

CLUT

The GLCDC module supports a Color Look-Up Table that is used if the color format is ARGB1555, CLUT8, CLUT4, or CLUT1. The CLUT API can update CLUT0/CLUT1 SRAM (implemented inside the GLCDC hardware) for each of the graphics foreground or background screens.

Note

Make sure to call the CLUT API if you select a color format that uses the CLUT, before using the display_api_t::start API; otherwise, CLUT0 and CLUT1 become an unknown condition and the graphics do not display properly.

You can also call the CLUT API at run-time to update CLUT SRAM.

Note

The API copies the source of CLUT data to the CLUT SRAM, which is not currently used (each CLUT SRAM consists of a ping-pong buffer). After completing the CLUT data update, the API automatically switches the CLUT SRAM to be read by the GLCDC hardware from the next frame to avoid tearing of the image.

Line Repeating Mode

Line Repeating is an important mode, especially for a system that does not have enough of memory. In this mode, the GLCDC module reads a raster image, which has fewer pixels than the LCD panel screen size, and displays the raster repeatedly on the screen. The following figure shows an example of a screen image constructed by reading a small raster image repeatedly in the background graphics plane.

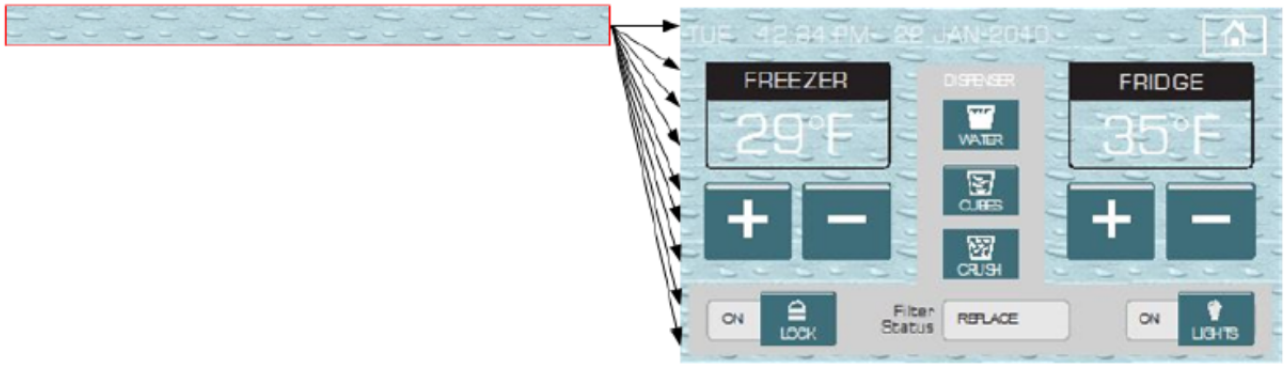


Figure 279: GLCDC HAL Module Line Repeating Mode

Note

To enable this mode, set the GLCDC module property "Input - Graphics screen N input lines repeat" (where N = 1 or 2) to ON with the Synergy configurator. Also, specify the repeat times to read a raster image to: "Input - Graphics screen N input lines repeat times". Specify the horizontal pixel size of the raster image in "Input - Graphics screen N input horizontal size" and "Input - Graphics screen N input horizontal stride," and then specify the vertical pixel size of the raster image in "Input - Graphics screen N input vertical size."

Gamma Correction

Gamma Correction is used to change the color characteristic of LCD panels to a flat characteristic. The following figure shows the gamma correction curve, which can be configured by the GLCDC module. The module supports 16 threshold values for the input color level for each (R, G, B) color and defines the gain level for each of 16 areas divided by thresholds.

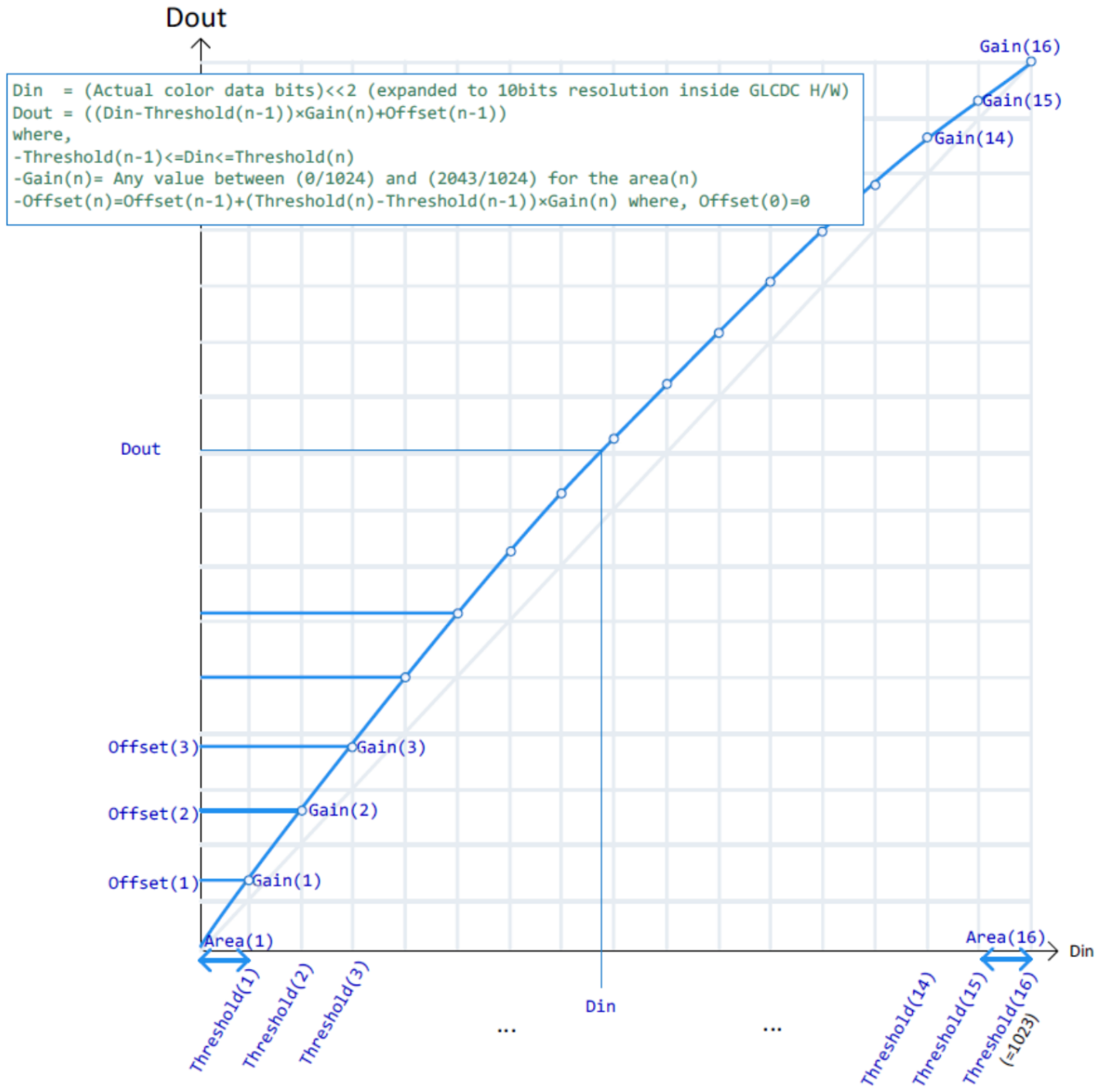


Figure 280: GLCDC HAL Module Gamma Correction Curve

Note

To enable the gamma correction for each channel (R, G, B), set the GLCDC module property "Color correction – Gamma correction (R, G, B)" to ON using the Synergy configurator. Thresholds (total 16) are set to "Color correction – Gamma correction threshold (R, G, B) [n]," where n=[0..15]. The gain value for each of areas are set to "Color correction – Gamma correction gain (R, G, B) \[n]," where n= [0..15].

GLCDC HAL Module Important Operational Notes and Limitations

GLCDC HAL Module Operational Notes

You have the option to configure multiple GLCDC interrupts covered in the following sections:

Line Detection Interrupt

The line detection interrupt is used to indicate when the GLCDC finishes outputting all lines to the LCD panel and goes into the blanking period. Use this interrupt to handle frame buffer switching in a graphics system and uses frame buffers with more than two frames.

Layer1 or Layer2 Line Buffer Underflow Interrupt

You can use the GLCDC layer1 or layer2 buffer underflow interrupt to detect lack of memory bandwidth in your system. The buffer underflow occurs when the graphics data transfer from memory (for example, SDRAM or SRAM) to the GLCDC internal line buffer is blocked by the other data transfer, and not enough against the data transfer from GLCDC line buffer to the LCD panel interface. You have to design the graphics system to prevent this interrupt from occurring.

GLCDC Callbacks

A user callback function can be registered in open. If a user callback function is provided, the callback function is called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function event can take the following enumerated value listed the table, so that a user can identify which event occurred in the graphics system. DISPLAY_EVENT_LINE_DETECTION event is used for switching frame buffers to update the screen, and DISPLAY_EVENT_GRn_UNDERFLOW event is used for error handling if an underflow occurs.

Event and Interrupt Summary

Name of Event	Name of Interrupt	Condition for the Event
DISPLAY_EVENT_LINE_DETECTION	Line detection	When GLCDC is done outputting the last line in the active video region
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics 1 underflow	When GLCDC underflows during reading the data for graphics1 plane
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics 2 underflow	When GLCDC underflows during reading the data for graphics2 plane

Note

Since the callback is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending an excessive time in an ISR can affect the responsiveness of the system.

GLCDC HAL Module Limitations

- The Display driver on r_glcd does not support RGB-index chroma key.
- The Display driver on r_glcd does not support the event link function.

4.2.14.4 Including the GLCDC HAL Module in an Application

This section describes how to include the GLCDC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and

configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Display Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Display Driver is g_display0. This name can be changed in the associated Properties window.)

GLCDC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_display0 Display Driver on r_glcdc	Threads	**New Stack> Driver> Graphics> Display Driver on r_glcdc**

When the Display Driver on r_glcdc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

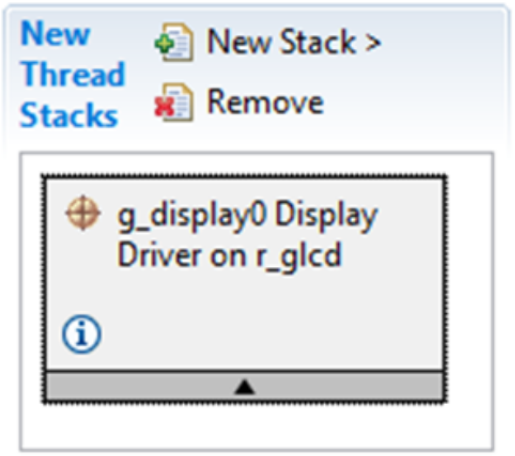


Figure 281: GLCDC HAL Module Stack

4.2.14.5 Configuring the GLCDC HAL Module

The GLCDC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the GLCDC HAL Module on r_glcdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_display0	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.
Input - Panel clock source select	Internal clock (GLCDCLK), External clock (LCD_EXTCLK) Default: Internal clock (GLCDCLK)	Choose the panel clock source depends on your system.
Input - Graphics screen1	Used, Not used Default: Used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.
Input - Number of Graphics screen1 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen1 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen1 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels

Input - Graphics screen1 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen1 input line descending	Used, Not used Default: Not used	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen1 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.

Input - Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Input - Graphics screen2	Used, Not used Default: Not used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen2 frame buffer name	fb_foreground	Custom name for frame buffer.
Input - Number of Graphics screen2 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."

Input - Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen2 input line descending	On, Off Default: Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen2 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.

Input - Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output - Horizontal total cycles	1024	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal active video cycles	800	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

Output - Horizontal back porch cycles	46	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal cycles	20	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal polarity	Low active, High active Default: Low active	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical total lines	525	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical active video lines	480	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical back porch lines	23	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

Output - Vertical sync signal lines	10	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal polarity	Low active, High active Default: Low active	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.
Output - Format	24bits RGB888, 18bits RGB666, 16bits RGB565, 8bits serial Default: 24bits RGB888	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Endian	Little endian, Big endian Default: Little endian	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Color order	RGB, BGR Default: RGB	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Data Enable Signal Polarity	Low active, High active Default: High active	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Sync edge	Rising Edge, Falling Edge Default: Rising Edge	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Background color alpha channel	255	Specify the background color of the background screens.
Output - Background color R channel	0	Specify the background color of the background screens.
Output - Background color G channel	0	Specify the background color of the background screens.
Output - Background color B channel	0	Specify the background color of the background screens.

CLUT	Used, Not used Default: Not used	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entries consumes 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.
TCON - Hsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Vsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON1	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - DataEnable pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Panel clock division ratio	1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/12, 1/16, 1/24, 1/32 Default: 1/8	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction - Brightness	Off, On Default: Off	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction - Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.

Color correction - Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Contrast	Off, On Default: Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.
Color correction - Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Gamma correction(Red)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).

Color correction - Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Green)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain G[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold G[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Blue)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.

Color correction - Gamma gain B[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold B[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Dithering	Off, On Default: Off	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Mode	Truncate, Round off, 2x2 Pattern Default: Truncate	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern A	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.

Dithering - Pattern B	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern C	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern D	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Misc - Correction Process Order	Brightness and Contrast then Gamma, Gamma then Brightness and Contrast Default: Brightness and Contrast then Gamma	Specify the color correction processing order if needed.
Line Detect Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.
Underflow 1 Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.
Underflow 2 Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

GLCDC HAL Module Clock Configuration

The GLCDC module can generate the pixel clock from either of the following clock sources (the source clock selection is available through Synergy Configuration in e^2 studio):

- Internal clock source (PLLOUT; 240 MHz)
- External clock source from the LCD_EXTCLK pin

Note

The internal clock is different in the S7G2 WS1 (Working Sample1) chip and the WS2 (Working Sample2) chip or later. The WS1 chip uses the PCLKB (max. 60 MHz), but WS2 or later chips use the PLLOUT (max. 240 MHz).

GLCDC HAL Module Pin Configuration

The GLCDC module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The pin selection table lists methods to select pins within the SSP configuration window and the configuration settings table lists an example depicting selection of GLCDC pins:

Pin Selection for the GLCDC HAL Module on r_glcdc

Resource	ISDE Tab	Pin selection Sequence
GLCDC	Pins	Select **Peripherals**> **Graphics: GLCDC**> **GLCDC0**

Note

The selection sequence assumes GLCDC0 is the desired hardware target for the driver.

Pin Configuration Settings for GLCDC HAL Module on r_glcd

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only Default: Mixed	Pin group selection.
Operation Mode	Disabled, Custom, RGB888, RGB666, RGB565 Default: Disabled	Select desired operation mode.
LCD_CLK	None, P900, P101 Default: None	LCD_CLK Pin.
LCD_DATA00:15	None, Pn, Pm Default: None	LCD_DATA Pins.
LCD_TCON0:3	None, Pn, Pm Default: None	LCD_TCON Pins.
LCD_EXTCLK	None, Pn, Pm Default: None	LCD_EXTCLK Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.
To use the GLCDC module on the S7G2 PE-HMI1 board, be sure to set up PORT10 pin3 (PA03) and pin5 (PA05) as IOPORT pins with the output level HIGH. Pin PA03 controls the DISP signal (Display on/off) and Pin PA05 controls the backlight of LCD panel. For details, see the schematics of S7G2 PE-HMI1 board.

4.2.14.6 Using the GLCDC HAL Module in an Application

- The typical steps in using the GLCDC HAL module in an application are:

1. Initialize the GLCDC HAL module with the display_api_t::open API.
2. Draw a primary image in the frame buffer with application code.
3. Start the image displaying using the display_api_t::start API.
4. Stop the image displaying using the display_api_t::stop API.
5. Copy the source CLUT data to CLUT SRAM using the display_api_t::clut API.

6. Draw a new image in the frame buffer to update the display with application code.

    - *Note*

      In a typical application a ping-pong frame buffer system is used, so the application will draw the image to another frame buffer, which is not used for displaying at this point.

7. Request frame buffer toggling to the GLCDC hardware with the display_api_t::layerChange API.

    - *Note*

      The GLCDC hardware toggles the frame buffer and displays a new image from the next frame.

8. Perform color correction using the display_api_t::correction API.
9. Get display status using the display_api_t::statusGet API.
10. Get the driver version using display_api_t::versionGet API. (Optional)
11. Close the GLCDC module by calling the display_api_t::close API.

To synchronize application code to the completion of drawing current frame buffer, use the Line Detection Interrupt and notify the timing to application code through the GLCDC callback.

These common steps are illustrated in a typical operational flow diagram in the following figure:

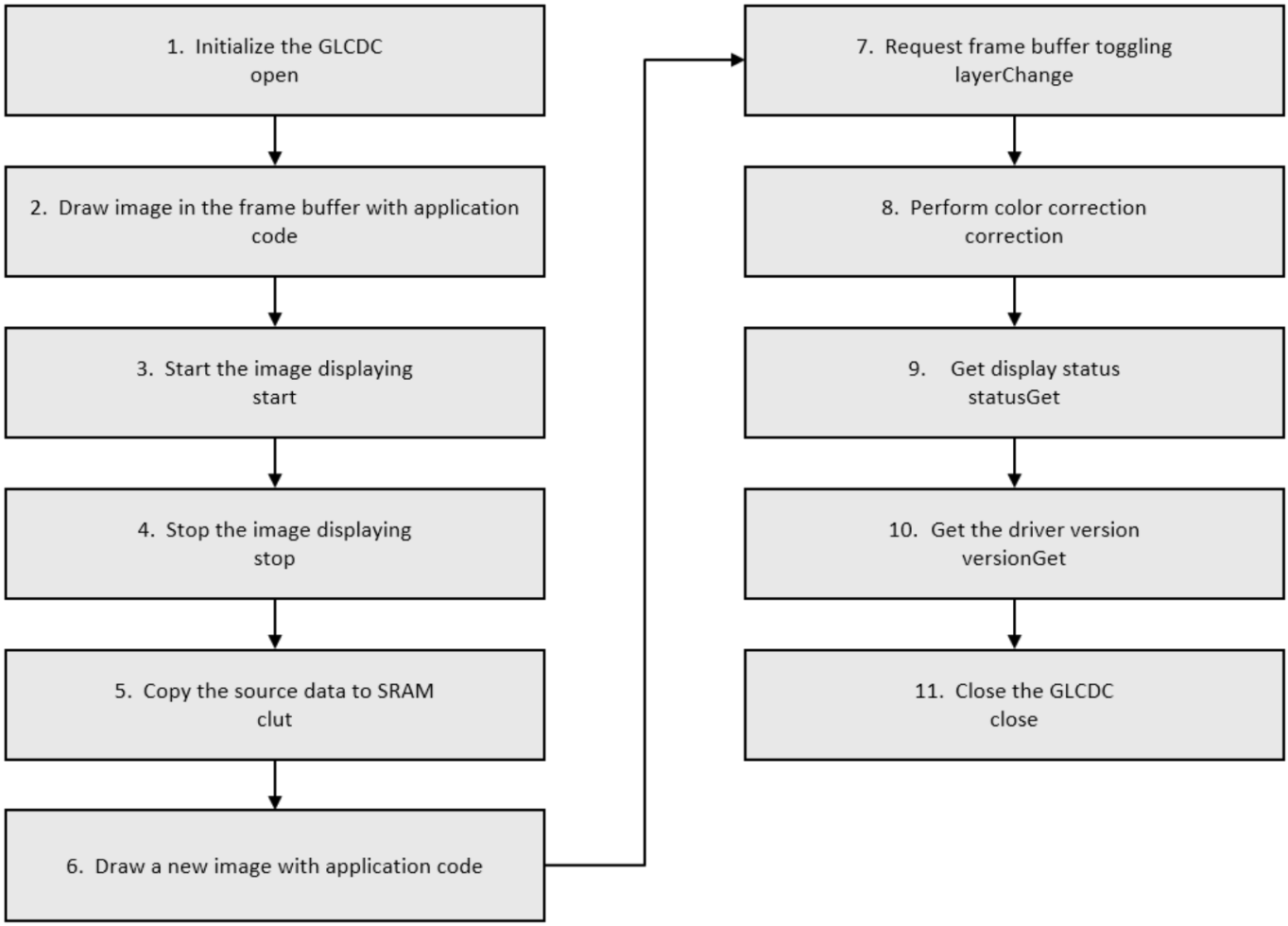


Figure 282: Flow Diagram of a Typical GLCDC HAL Module Application

4.2.15 Data Operation Circuit Driver

4.2.15.1 DOC HAL Module Introduction

The Data Operation Circuit (DOC) HAL module provides a high-level API for DOC applications and uses the DOC peripherals on the Synergy MCU. A user-defined callback can be created to inform the CPU when an event occurs.

DOC HAL Module Features

The DOC HAL module peripheral is used to compare 16-bit data and can detect the following events:

- A mismatch or match between data values
- Overflow of an addition operation
- Underflow of a subtraction operation

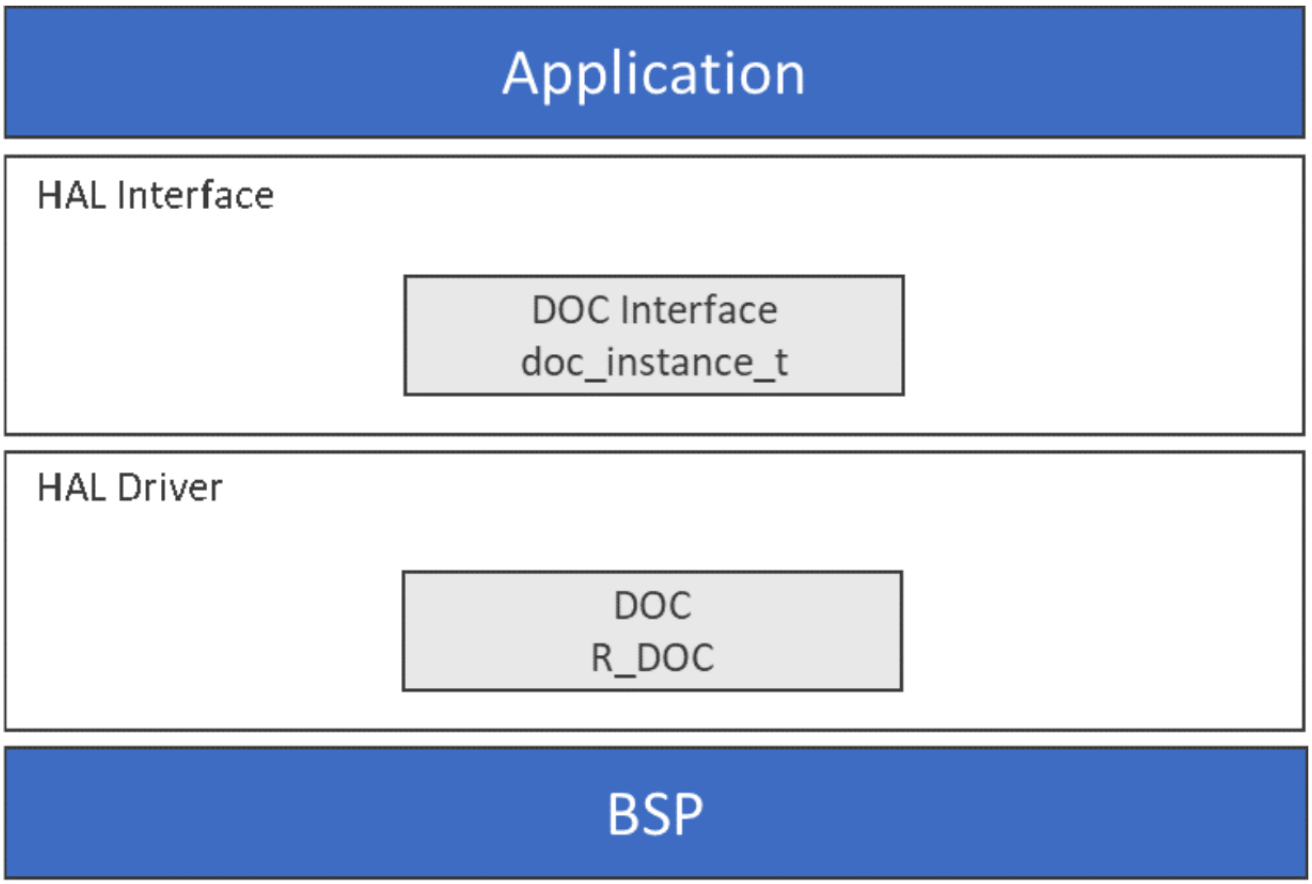


Figure 283: DOC HAL Module Block Diagram

DOC Hardware support details

The following hardware features are, or are not, supported by SSP for DOC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Module-stop Function	Event Link Function through ELC HAL Driver
S124	✓	☒
S128	✓	☒
S1JA	✓	☒
S3A1	✓	☒
S3A3	✓	☒
S3A6	✓	☒
S3A7	✓	☒
S5D3	✓	☒
S5D5	✓	☒
S5D9	✓	☒
S7G2	✓	☒

4.2.15.2 DOC HAL Module APIs Overview

The DOC HAL module defines APIs for opening, closing, checking the status of and writing data to the data operation circuit. The DOC HAL module uses the DOC peripheral on the Synergy MCU. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

DOC HAL Module API Summary

Function Name	Example API Call and Description
open	g_doc.p_api->open(g_doc.p_ctrl, g_doc.p_cfg) Initial configuration.
close	g_doc.p_api->close(g_doc.p_ctrl) Allow the driver to be reconfigured. Will reduce power consumption.
statusGet	g_doc.p_api->statusGet(g_doc.p_ctrl, &my_Status) Get the DOC status and stores it in the provided pointer p_status.
statusClear	g_doc.p_api->statusClear(g_doc.p_ctrl) Clear DOPCF status flag.
write	g_doc.p_api->write(g_doc.p_ctrl, value) Write to the DODIR and DODSR registers.

inputRegisterWrite	g_doc.p_api->inputRegisterWrite(g_doc.p_ctrl, &doc_values) Write to the DODIR register.
versionGet	g_doc.p_api->versionGet(g_doc.p_ctrl, &version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	DOC successfully configured.
SSP_ERR_IN_USE	Module already open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_INVALID_ARGUMENT	ISR is not enabled. Enable the ISR in bsp_irq_cfg.h.
SSP_ERR_HW_LOCKED	DOC resource is locked.
SSP_ERR_NOT_OPEN	Driver not open.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.15.3 DOC HAL Module Operational Overview

The DOC HAL module controls the DOC peripheral on a Synergy MCU. It is used to compare 16-bit data and can detect a mismatch between data values, an overflow of an addition value, or an underflow of a subtraction operation. If a callback is available and the associated interrupt is enabled, the callback function will be called in response to a DOC event.

The DOC uses two data registers to perform operations: the DOC Data Input Register (DOCDIR) holds the data to be operated on and the DOC Data Setting Register (DOCDSR) holds the value that is used to operate on the input data. In the addition and subtraction modes, this register stores the results of data operations. (Both these registers are 16-bits wide.)

DOC HAL Module Important Operational Notes and Limitations

DOC HAL Module Operational Notes

The initial setting of comparison data is written to the DOC by calling the doc_api_t::write API. The doc_api_t::write API writes to the DOC DODSR and DODIR registers. The doc_api_t::write API use a variable of type doc_data_t, as illustrated below:

```
doc_data_t g_doc_values;

g_doc_values.dodir = 0x1000;
```

```
g_doc_values.dodsr = 0x1000;

g_doc.p_api->write(g_doc.p_ctrl, &g_doc_values);
```

If the data to be compared does not change, there is no need to re-write it each time a comparison is required. The input data value can be written to the DOC by using the doc_api_t::inputRegisterWrite API. The doc_api_t::inputRegisterWrite API writes only to the DOC data-input register.

DOC HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.15.4 Including the DOC HAL Module in an Application

This section describes how to include the DOC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the DOC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the DOC Driver is g_doc0. This name can be changed in the associated Properties window.)

DOC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_doc0 Data Operation Circuit Driver on r_doc	Threads	**New Stack> Driver> Monitoring> Data Operation Circuit Driver on r_doc**

When the Key Matrix Driver on r_kint is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

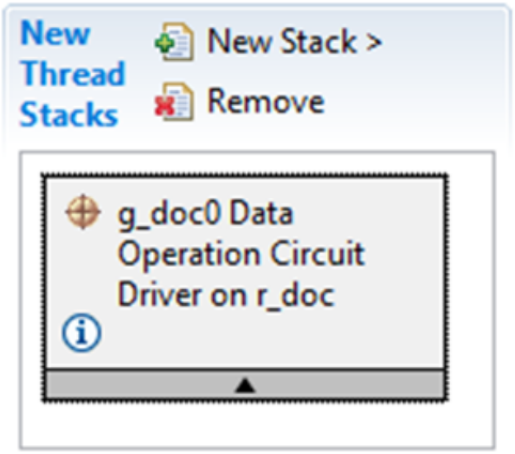


Figure 284: DOC HAL Module Stack

4.2.15.5 Configuring the DOC HAL Module

The DOC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DOC HAL Module on r_doc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_doc0	Module name.
Event	Comparison mismatch, Comparison match, Addition overflow, Subtraction underflow Default: Comparison mismatch	Specify the event which will trigger the DOC interrupt.

Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) when the configured DOC event occurs. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
DOC Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Use the pull down to set the DOC interrupt priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DOC HAL Module Clock Configuration

The DOC HAL module does not require a specific clock configuration.

DOC HAL Module Pin Configuration

The DOC HAL module does not require and specific pin configurations.

4.2.15.6 Using the DOC HAL Module in an Application

The typical steps in using the DOC HAL module in an application are:

1. Initialize the DOC using the doc_api_t::open API.
2. Set register values in DODIR and DODSR using the doc_api_t::write API.
3. Steam data to the DOC using the doc_api_t::inputRegisterWrite API.
4. Read the status of the comparison using the doc_api_t::statusGet API or in the callback if enabled.
5. Clear status flags using the doc_api_t::statusClear API.
6. Close the module using the doc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

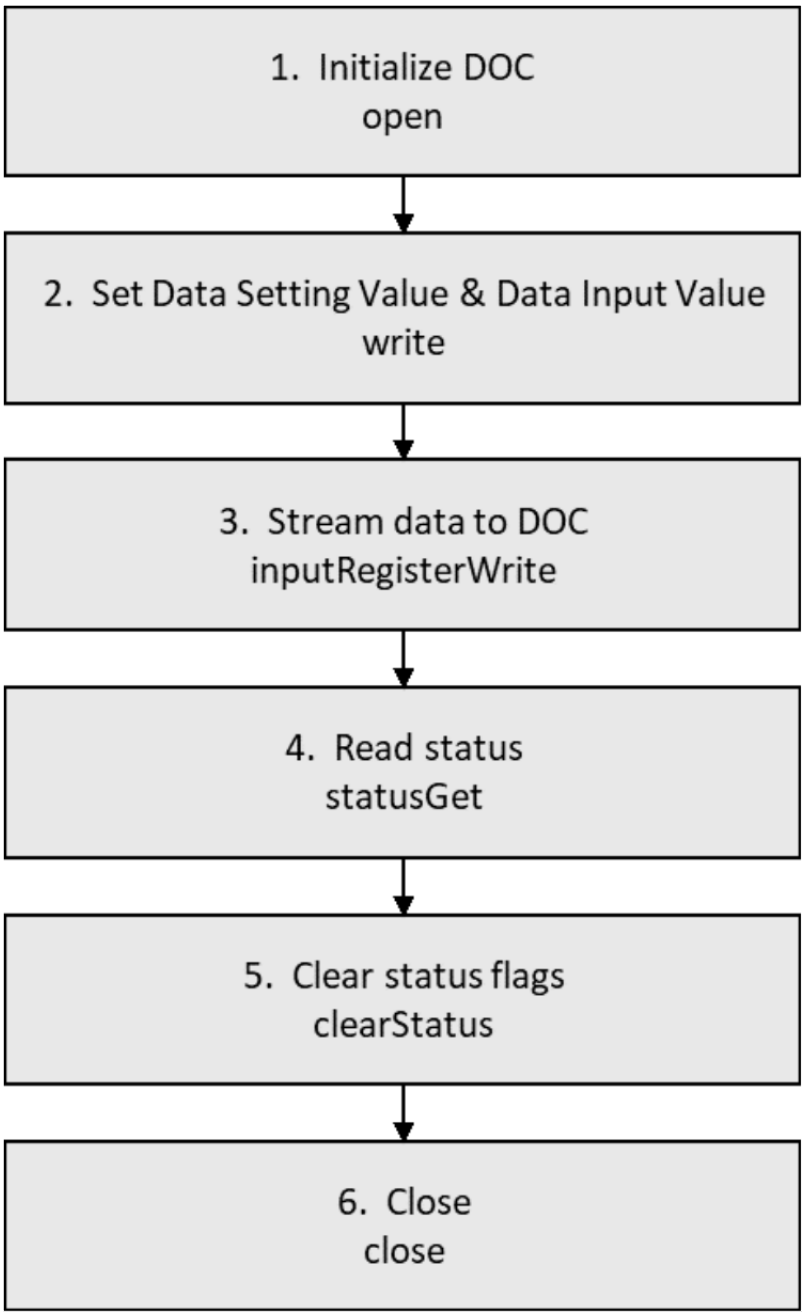


Figure 285: Flow Diagram of a Typical DOC HAL Module Application

4.2.16 Transfer Driver on r_dmac

4.2.16.1 DMAC HAL Module Introduction

The DMAC HAL module provides a high-level API for data-transfer applications and uses the DMAC peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU when transfer events occur.

DMAC HAL Module Features

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module supports the following:

- DMAC module on a Synergy MCU
- Interrupts, if desired
- Multiple transfer modes
    - Single Transfer
    - Repeat Transfer
    - Block Transfer
    - Address increment, decrement, offset addition, or fixed modes
- Multiple channels, with the number depending on the MCU used

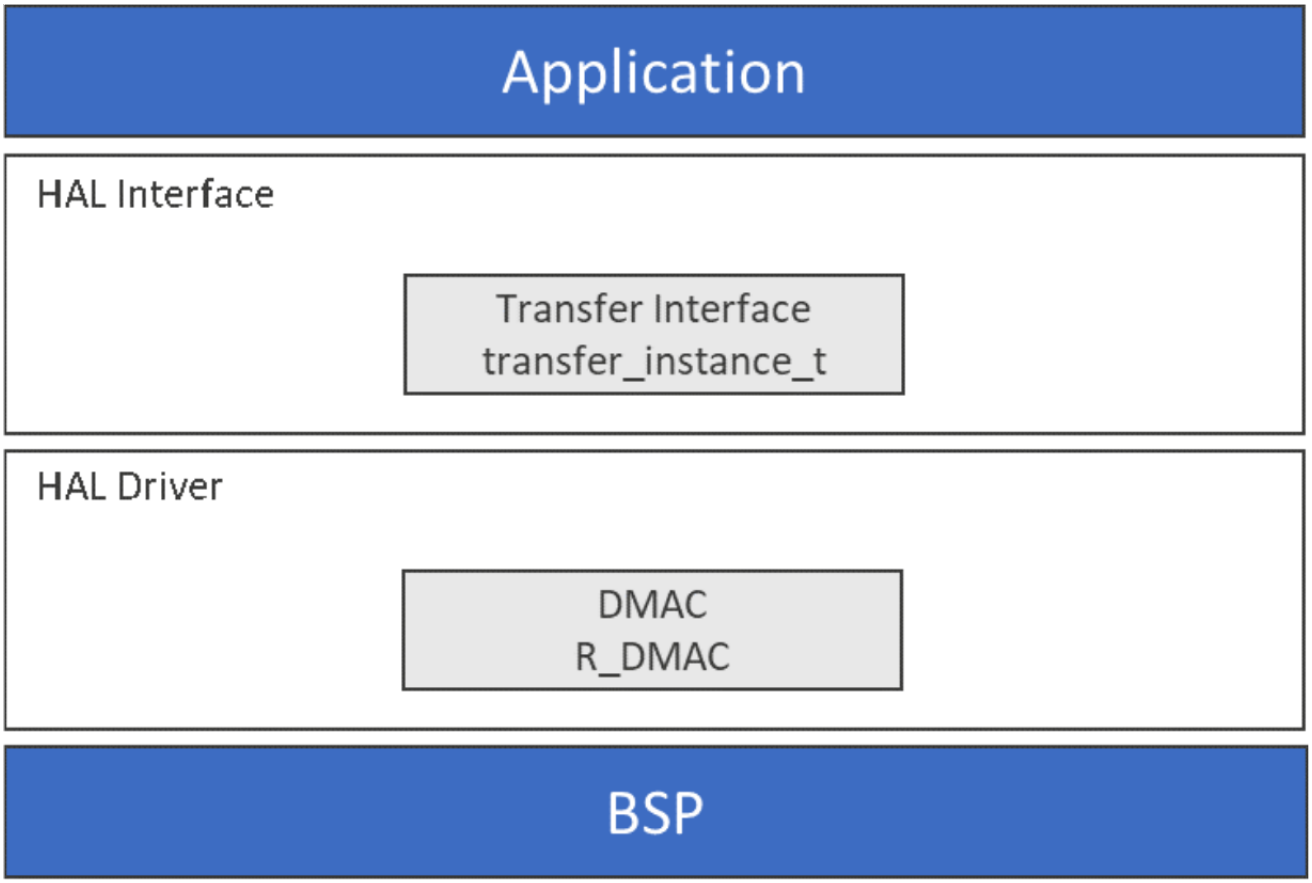


Figure 286: DMAC HAL Module Block Diagram

DMAC Hardware support details

The following hardware features are, or are not, supported by SSP for DMAC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A	Not supported by MCU

MCU Group	Normal Transfer Mode	Repeat Transfer Mode	Block Transfer Mode	Extended repeat area function	Event link function through ELC HAL driver
S124	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A
S3A1	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

4.2.16.2 DMAC HAL Module APIs Overview

The DMAC HAL module defines APIs for opening, closing, starting and stopping timers. Note that the Data Transfer Controller (DTC) and the DMAC use the same transfer interface; sharing an interface makes it easier to change between DTC and DMA implementations. The API calls are the same independent of the lower level implementations. A complete list of the available APIs, an example API call and a short description of each function can be found in the following table. A table of status return values follows the API summary table.

DMAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_transfer0.api->open(g_transfer0.p_ctrl, g_transfer0.p_cfg) Open device channel. Initialize driver and hardware on first call.
close	g_transfer0.api->close(g_transfer0.p_ctrl) Close device channel. Turns off hardware if last channel open.
reset	g_transfer0.api->reset(g_transfer0.p_ctrl, &source, &destination, number_of_transfers) Reset channel settings.

start	g_transfer0.api->start(g_transfer0.p_ctrl, mode) Start data transfer.
stop	g_transfer0.api->stop(g_transfer0.p_ctrl) Stop data transfer.
enable	g_transfer0.api->enable(g_transfer0.p_ctrl) Enable channel.
disable	g_transfer0.api->disable(g_transfer0.p_ctrl) Disable channel.
versionGet	g_transfer0.api->versionGet(&version) Retrieve the API version with the version pointer.
infoGet	g_transfer0.api->infoGet(g_transfer0.p_ctrl, &info) Get transfer channel info.
blockReset	g_transfer0.api->blockReset(g_transfer0.p_ctrl, &source, &destination, length, size, number_of_transfers) Reset Block Transfer parameters.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNSUPPORTED	Operation not configured correctly.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
SSP_ERR_NOT_ENABLED	Operation failed.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INVALID_SIZE	Invalid Offset Value.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.16.3 DMAC HAL Module Operational Overview

The DMAC and the DTC can be used to move data within the Synergy MCU. There are a few considerations when selecting between these modules and you need to determine which implementation works best for your application. The DTC module is recommended for most generic transfer applications and is also available for data transfer operations. The following use cases describe operations within each transfer module.

Selecting the DTC Transfer Module

The DTC HAL module uses a RAM based vector table, with slots for every interrupt in the system. When the DTC transfer completes, the activation source interrupt is called. The activation source interrupt must be enabled to use the DTC. The activation source interrupt is generally muted by the DTC until the transfer completes, unless TRANSFER_IRQ_EACH is specified in the configuration. For example, if a normal mode transfer with a length of 16 is triggered by a timer, the timer interrupt does not fire the first 15 times while the transfer is in effect. After the 16th transfer, the timer interrupt fires. The DTC also allows chained transfers, meaning that more than one transfer can occur after a single activation source interrupt. This feature is supported by the driver but must be configured outside the ISDE.

Selecting the DMAC Transfer Module

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module uses DMAC peripheral registers, so the number of transfers in the system is limited to the number of DMAC channels on the device. The activation source does not have to be enabled to use the DMAC. When the DMAC transfer completes, a DMAC interrupt is called. If the activation source interrupt is enabled, it fires at the same time the transfer is triggered. If the DMAC interrupt is enabled, it fires after all transfers are complete. For example, if a normal mode transfer, with a length of 16 is triggered by a timer, the timer interrupt fires at the same time each transfer occurs and the DMAC interrupt fires after the 16th transfer completes. The DMAC does not support chained transfers.

DMAC HAL Module Important Operational Notes and Limitations

DMAC HAL Module Operational Notes

Normal Mode

In normal mode, a single transfer triggers each time an activation source event occurs. A single transfer is 1 byte, 2 bytes or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length decrements by 1. When the transfer length reaches 0, the transfer is complete.

Repeat Mode

In repeat mode, a single transfer triggers each time an activation source event occurs. A single transfer is 1 byte, 2 bytes or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length decrements by 1. When transfer length reaches 0, the transfer length reloads with its initial value and the transfer restarts. If the repeat area is set to source, the source register also reloads with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register reloads with its initial value when the transfer restarts.

Block Mode

In block mode, the entire transfer length transfers each time an activation source event occurs. For example, if a transfer is configured in the block mode with a timer as the activation source, a 2-byte

size and 12-byte length, 24 bytes transfer each time the activation source event occurs. Each time a transfer occurs, the transfer length decrements by 1. When the length reaches 0, the transfer length reloads with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register reloads with its initial value when the transfer restarts.

Address Mode

After each transfer of size (1 byte, 2 bytes, or 4 bytes), the source pointer and destination pointer adjust by transfer_info_t::src_addr_mode and transfer_info_t::dest_addr_mode, respectively.

For example, if transfer_info_t::src_addr_mode is set to TRANSFER_ADDR_MODE_INCREMENTED, and size is set to TRANSFER_SIZE_4_BYTE, the transfer_info_t::p_dest pointer is incremented by 4 (the transfer size) after each transfer.

For TRANSFER_ADDR_MODE_OFFSET, the pointer is incremented or decremented by the configured offset value.

The pointer does not change if set to TRANSFER_ADDR_MODE_FIXED.

DMAC HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.16.4 Including the DMAC HAL Module in an Application

This section describes how to include the DMAC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Transfer Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Transfer Driver is g_dmac0. This name can be changed in the associated Properties window.)

DMAC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ transfer0 Transfer Driver on r_dmac	Threads	**New Stack> Driver> Transfer> Transfer Driver on r_dmac**

When the Transfer Driver on r_dmac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and

displays possible choices.

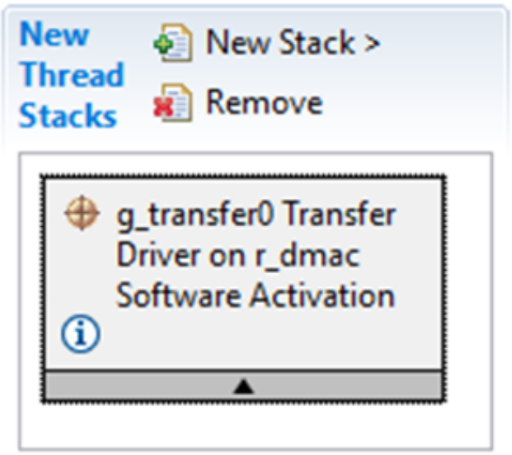


Figure 287: DMAC HAL Module Stack

4.2.16.5 Configuring the DMAC HAL Module

The DMAC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DMAC HAL Module on r_dmac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer0	Module name.
Channel	0	Channel selection.
Mode	Normal, Repeat, Block Default: Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes Default: 2 Bytes	Transfer size selection.

Destination Address Mode	Fixed, Incremented, Destination Default: Fixed	Destination address mode selection.
Source Address Mode	Fixed, Incremented, Destination Default: Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination, Source Default: Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Offset Addition (Valid only in Offset Addition mode)	-16777216 to 16777215 Default: 0	Offset value selection.
Activation Source	Software Activation, Peripheral Events Default: Software Activation	Activation source selection.
Auto Enable	True, False Default: True	Auto enable selection.
Callback	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DMAC HAL Module Clock Configuration

The DMAC peripheral module use ICLK as the clock source. The ICLK frequency is set by using the SSP configurator clock tab, prior to a build, or by using the CGC Interface at run-time.

DMAC HAL Module Pin Configuration

The DMAC HAL Module is not associated with any pins.

4.2.16.6 Using the DMAC HAL Module in an Application

The typical steps in using the DMAC HAL module in an application are:

1. Initialize the DMAC HAL module using the transfer_api_t::open API.
2. Enable the DMAC HAL module using the transfer_api_t::enable API (if not auto enabled).
3. Manage transfers using other APIs as needed.
4. Close the DMAC HAL module using the transfer_api_t::close API when needed.

These common steps are illustrated in a typical operational flow diagram in the following figure:

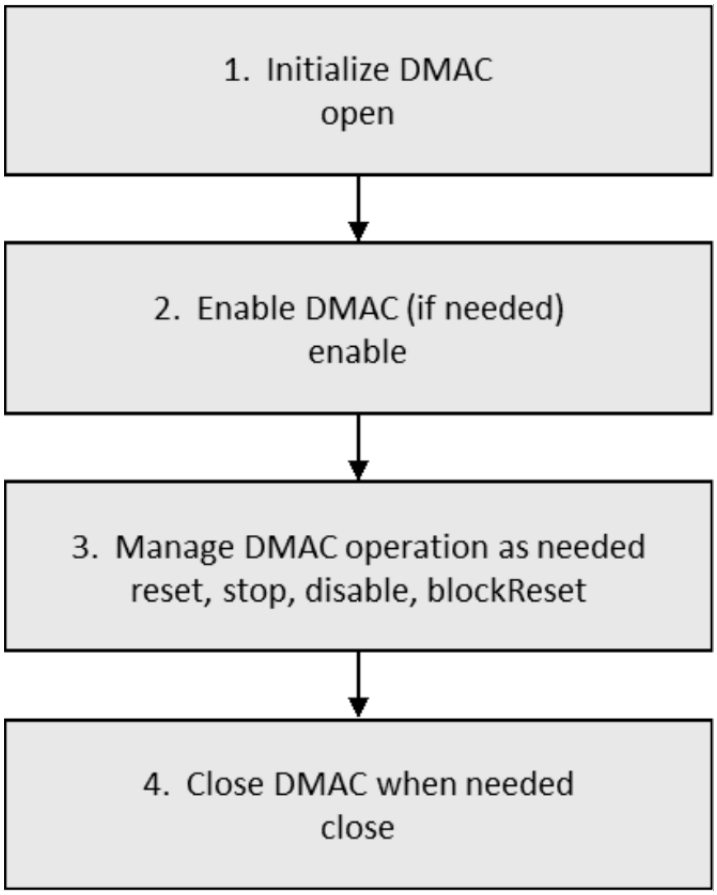


Figure 288: Flow Diagram of a Typical DMAC HAL Module Application

4.2.17 Transfer Driver on r_dtc

4.2.17.1 DTC HAL Module Introduction

The Data Transfer Controller (DTC) HAL module provides a high-level API for data-transfer applications and uses the DTC peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU when transfer events occur.

DTC HAL Module Features

The Data Transfer Controller (DTC) HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs.

- Supports the DTC module on a Synergy MCU
- Supports interrupts if desired
- Supports multiple transfer modes
    - Single transfer
    - Repeat transfer
    - Block transfer
    - Address increment or fixed modes
    - Chain transfers
- Supports multiple channels (depending on selected implementation)
    - Number of channels is limited only by the size of the RAM-based vector table

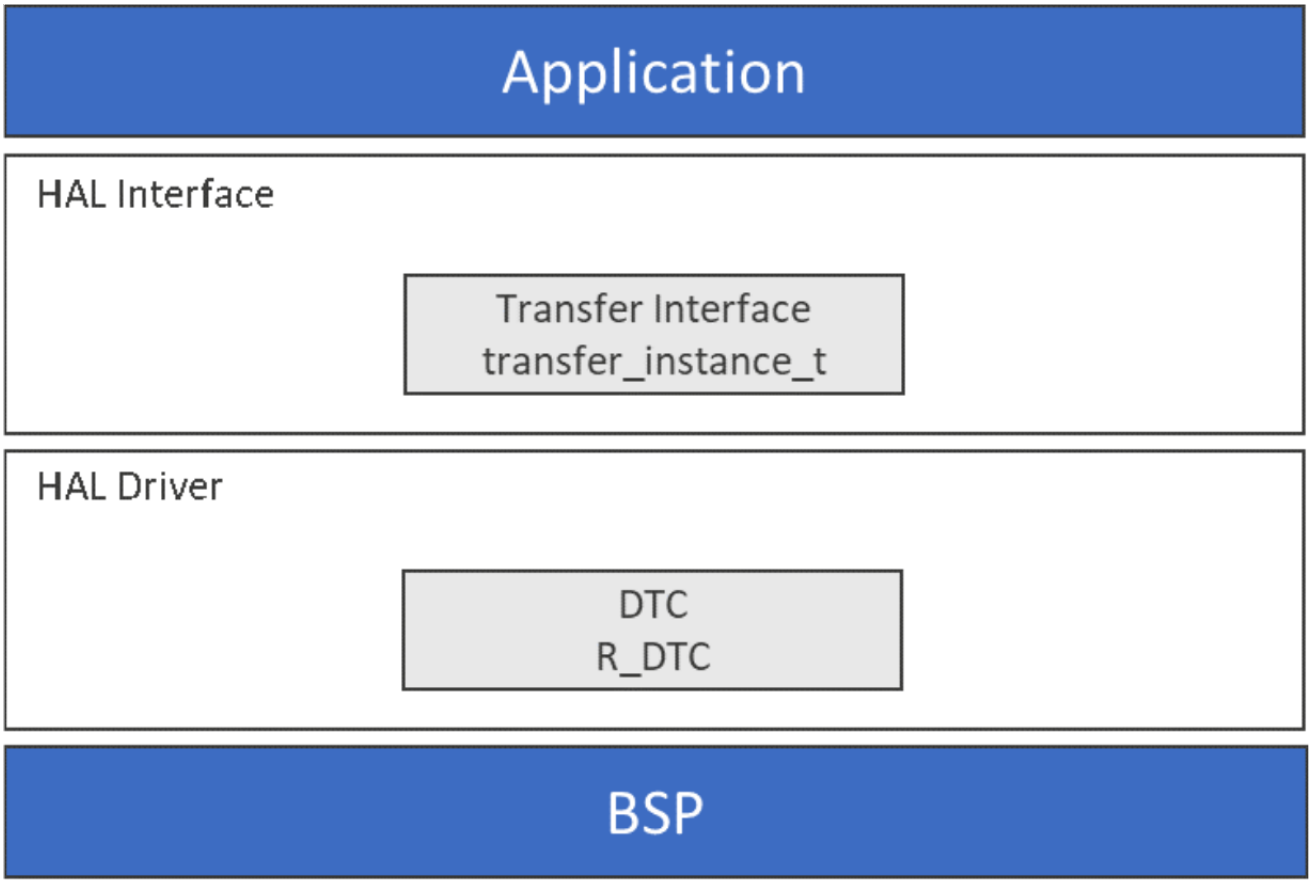


Figure 289: DTC HAL Module Block Diagram

DTC Hardware support details

The following hardware features are, or are not, supported by SSP for DTC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Normal Transfer Mode	Repeat Transfer Mode	Block Transfer Mode	Selectable Data Transfer Units (8 bits, 16 bits, 32 bits)	API/Config for Event link function upon completion of DTC transfers
S124	✓	✓	✓	✓	☒
S128	✓	✓	✓	✓	☒
S1JA	✓	✓	✓	✓	☒
S3A1	✓	✓	✓	✓	☒
S3A3	✓	✓	✓	✓	☒
S3A6	✓	✓	✓	✓	☒
S3A7	✓	✓	✓	✓	☒
S5D3	✓	✓	✓	✓	☒
S5D5	✓	✓	✓	✓	☒
S5D9	✓	✓	✓	✓	☒
S7G2	✓	✓	✓	✓	☒

4.2.17.2 DTC HAL Module APIs Overview

The DTC HAL module defines APIs for opening, closing, resetting, enabling, disabling, starting and stopping. Note that the DTC and the DMAC use the same transfer interface to make it easier to change between DTC and DMA implementations. The API calls are the same independent of the lower-level implementations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

DTC HAL Module API Summary

Function Name	Example API Call and Description
open	g_transfer0.api->open(g_transfer0.p_ctrl, g_transfer0.p_cfg) Initial configuration. Enables the transfer if auto_enable is true and p_src, p_dest, and length are valid. Transfers can also be enabled using enable or reset.
close	g_transfer0.api->close(g_transfer0.p_ctrl) Close device channel. Turns off hardware if last channel open.

reset	g_transfer0.api->reset(g_transfer0.p_ctrl, &source, &destination, number_of_transfers) Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.
start	g_transfer0.api->start(g_transfer0.p_ctrl, mode) Start transfer in software.
stop	g_transfer0.api->stop(g_transfer0.p_ctrl) Stop transfer in software. The transfer will stop after completion of the current transfer.
enable	g_transfer0.api->enable(g_transfer0.p_ctrl) Enable transfer. Transfers occur after the activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as activation source).
disable	g_transfer0.api->disable(g_transfer0.p_ctrl) Disable transfer. Transfers do not occur after the transfer_info_t::activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source).
versionGet	g_transfer0.api->versionGet(&version) Gets version and stores it in provided pointer version.
infoGet	g_transfer0.api->infoGet(g_transfer0.p_ctrl, &info) Provides information about this transfer.
blockReset	g_transfer0.api->blockReset(g_transfer0.p_ctrl, &source, &destination, length, size, number_of_transfers) Reset source address pointer, destination address pointer, and/or length, for block transfer keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.

SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNSUPPORTED	Operation not configured correctly.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_HW_LOCKED	The DTC hardware resource is locked.
SSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
SSP_ERR_NOT_ENABLED	Operation failed.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.17.3 DTC HAL Module Operational Overview

The Direct Memory Access Controller (DMAC) and the Data Transfer Controller (DTC) can be used to move data within the Synergy MCU. There are some considerations when selecting between these implementations; the following operational overview includes information on each to help you determine which implementation is best for your application. The DTC module is recommended for most generic transfer applications, but either module can be used for basic transfer functionality. The use-cases for each transfer module are:

Selecting the DMAC HAL Module

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module uses DMAC peripheral registers, so the number of transfers in the system is limited to the number of DMAC channels on the device. The activation source does not have to be enabled to use the DMAC. When the DMAC transfer completes, a DMAC interrupt is called. If the activation source interrupt is enabled, it fires at the same time the transfer is triggered. If the DMAC interrupt is enabled, it fires after all transfers are complete. For example, if a normal-mode transfer with a length of 16 is triggered by a timer, the timer interrupt fires at the same time each transfer occurs and the DMAC interrupt fires after the 16th transfer completes. The DMAC HAL module does not support chained transfers.

Selecting the DTC HAL Module

The DTC HAL module uses a RAM-based vector table with slots for every interrupt in the system. When the DTC transfer completes, the activation source interrupt is called. The activation source interrupt must be enabled to use the DTC. The activation source interrupt is generally muted by the DTC until the transfer completes, unless TRANSFER_IRQ_EACH is specified in the configuration. For example, if a normal-mode transfer with a length of 16 is triggered by a timer, the timer interrupt does not fire the first 15 times while the transfer is in effect. After the 16th transfer, the timer interrupt fires. The DTC also allows chained transfers, meaning that more than one transfer can occur after a single activation-source interrupt. This feature is supported by the driver but must be configured outside the ISDE.

DTC HAL Module Important Operational Notes and Limitations

DTC HAL Module Operational Notes

Normal Mode

In normal mode, a single transfer is triggered each time an activation-source event occurs. A single transfer is 1 byte, 2 bytes or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer is complete.

Repeat Mode

In repeat mode, a single transfer is triggered each time an activation-source event occurs. A single transfer is 1 byte, 2 bytes or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Block Mode

In the block mode, the entire transfer length is transferred each time an activation-source event occurs. For example, if a transfer is configured in block mode with the timer as the activation source, a 2-byte size, and a 12-byte length, 24 bytes are transferred each time the activation source event occurs. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Address Mode

After each transfer of size (1 byte, 2 bytes, or 4 bytes), the source pointer and destination pointer are adjusted based on the configuration settings for Source Address Mode and Destination Address Mode, respectively. For example, if the Source Address Mode is set to TRANSFER_ADDR_MODE_INCREMENTED and the size is set to TRANSFER_SIZE_4_BYTES, the destination pointer is incremented by 4 (the transfer size) after each transfer. The pointer does not change if the configuration setting is TRANSFER_ADDR_MODE_FIXED.

Chained Transfers

Chained transfers are only supported by the DTC. To use a chained transfer, create an array of transfer_info_t structures. Configure transfer_chain_mode_t to TRANSFER_CHAIN_MODE_EACH or TRANSFER_CHAIN_MODE_END for all transfers except the last transfer. The last transfer must be TRANSFER_CHAIN_MODE_DISABLED.

Set transfer_cfg_t::p_info to the base of the first structure in the array for transfer_info_t structures.

DTC HAL Module Limitations

- Refer to the most SSP Release Notes for any additional operational limitations for this module.

4.2.17.4 Including the DTC HAL Module in an Application

This section describes how to include the DTC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Transfer Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Transfer Driver is g_transfer0. This name can be changed in the associated Properties window.)

DTC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ transfer0 DTC Driver on r_dtc	**Threads**> **HAL/Common**	**New Stack**> **Driver**> **Transfer**> **Transfer Driver on r_dtc**

When the Transfer Driver on r_dtc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

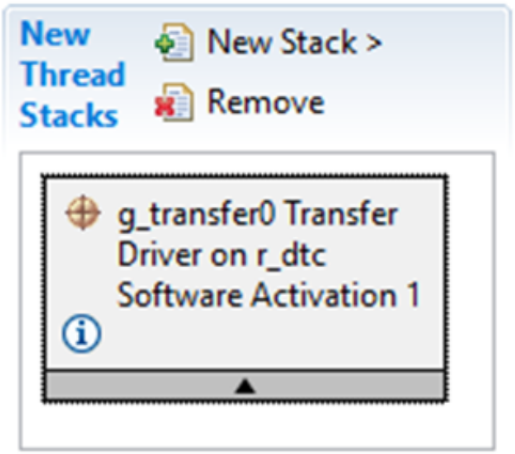


Figure 290: DTC HAL Module Stack

4.2.17.5 Configuring the DTC HAL Module

The DTC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties

tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DTC HAL Module on r_dtc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal, Repeat, Block Default: Normal	Mode selection.
Transfer Size	2 Bytes	Transfer size selection.
Destination Address Mode	Fixed, Incremented, Destination Default: Fixed	Destination address mode selection.
Source Address Mode	Fixed, Incremented, Destination Default: Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination, Source Default: Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.

Auto Enable	True, False Default: True	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DTC HAL Module Clock Configuration

The DTC peripheral module use ICLK as the clock source. The ICLK frequency is set by using the SSP configurator **Clocks** tab prior to a build, or by using the CGC Interface at run-time.

DTC HAL Module Pin Configuration

The DTC is not associated with any pins.

4.2.17.6 Using the DTC HAL Module in an Application

The typical steps in using the DTC HAL module in an application are:

1. Initialize the DTC using the transfer_api_t::open API.
2. Enable the DTC using the transfer_api_t::enable API (if not auto enabled).
3. Manage transfers using other APIs as needed.
4. Close the DTC with the transfer_api_t::close when needed.

These common steps are illustrated in a typical operational flow diagram in the following figure:

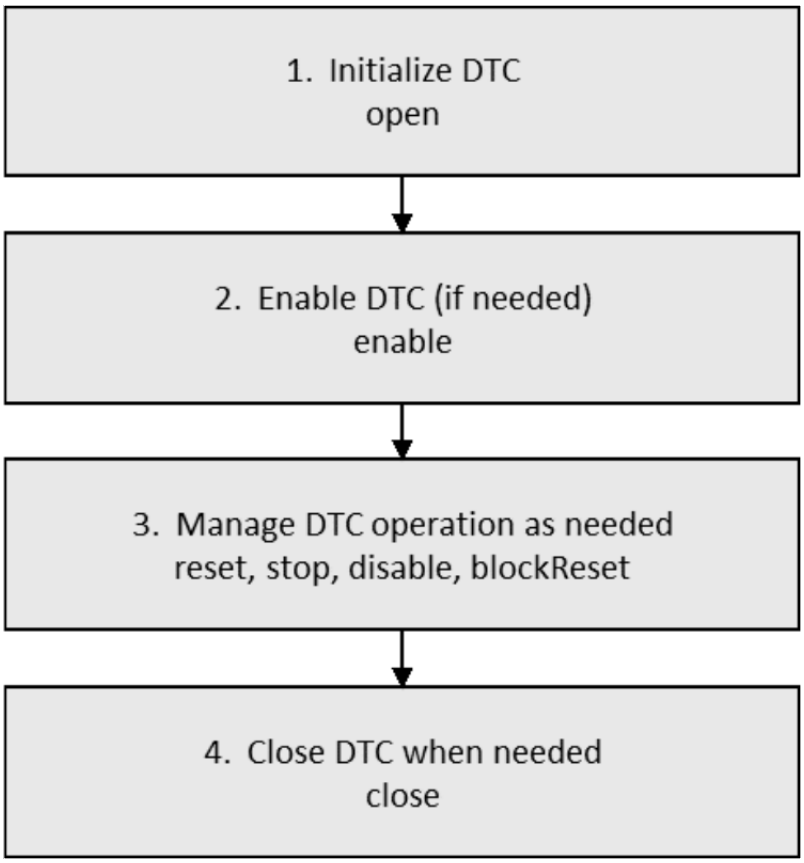


Figure 291: Flow Diagram of a Typical DTC HAL Module Application

4.2.18 ELC Driver

4.2.18.1 ELC HAL Module Introduction

The Event Link Controller (ELC) HAL module provides a high-level API for connecting various MCU peripherals for autonomous operation and uses the ELC peripheral on the Synergy MCU. There are no callbacks associated with the ELC HAL module. The project configurator in the e^2 studio Integrated Solution Development Environment (ISDE) includes the ELC HAL module in every project by default. To configure the ELC HAL module, select it in the HAL/Common module in the **Threads** tab and click on it in the HAL/Common Stacks window.

ELC HAL Module Features

The ELC HAL module can perform the following functions:

- Creates an event link between two blocks.
- Breaks that event link between two blocks.
- Generates one of two software events that interrupt the CPU.

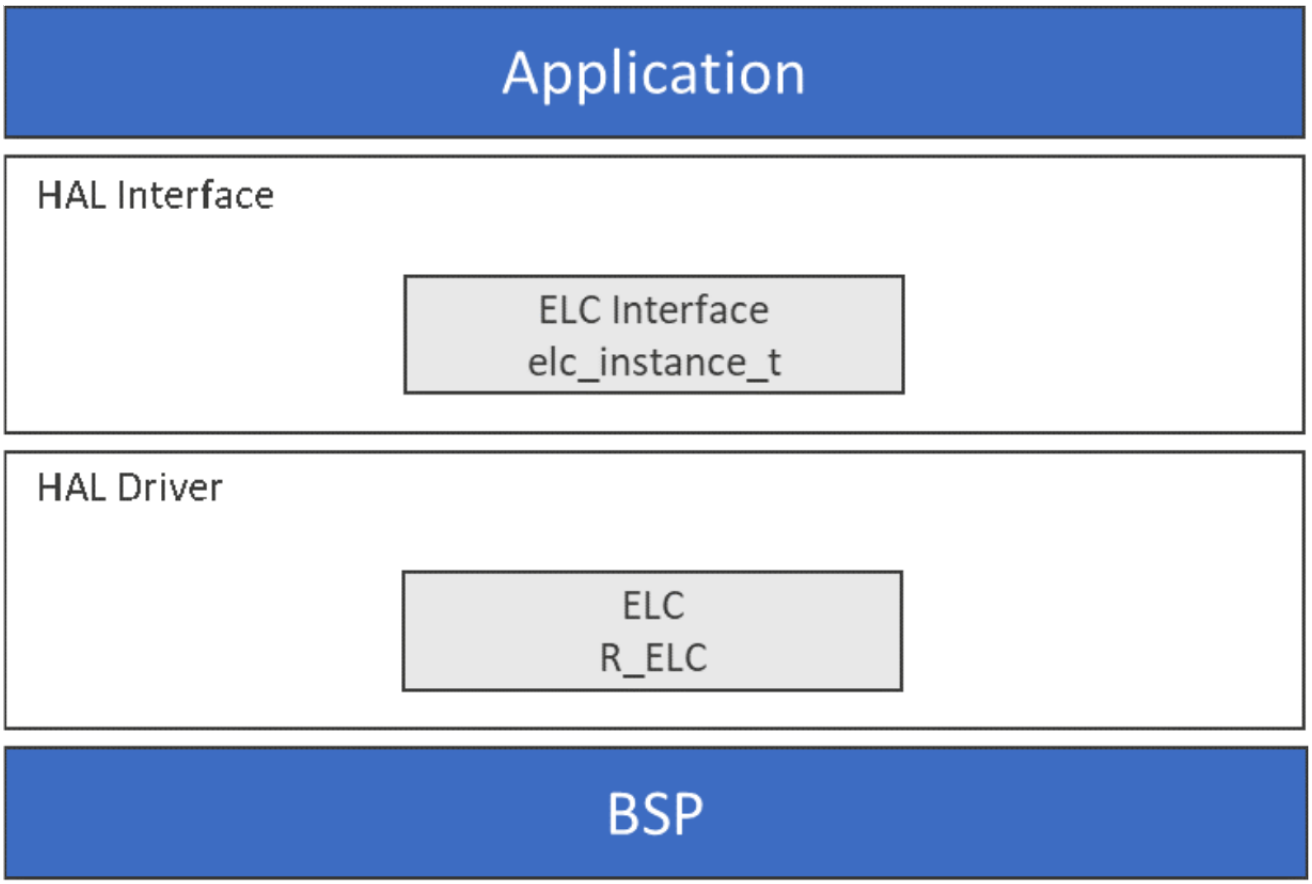


Figure 292: ELC HAL Module Block Diagram

ELC Hardware support details

The following hardware features are, or are not, supported by SSP for ELC.

Legend:

Symbol		Meaning	
✓		Available (Tested)	
☒		Not Available (Not tested/not functional or both)	
N/A		Not supported by MCU	
MCU Group	Create	Break	Generate
S124	✓	✓	✓
S128	✓	✓	✓
S1JA	✓	✓	✓
S3A1	✓	✓	✓
S3A3	✓	✓	✓
S3A6	✓	✓	✓

S3A7	✓	✓	✓
S5D3	✓	✓	✓
S5D5	✓	✓	✓
S5D9	✓	✓	✓
S7G2	✓	✓	✓

4.2.18.2 ELC HAL Module APIs Overview

The ELC HAL module defines APIs for initializing, enabling, disabling and creating or breaking event links between modules. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ELC HAL Module API Summary

Function Name	Example API Call and Description
init	g_elc.p_api->init(g_elc.p_cfg) Initialize all links in the Event Link Controller.
softwareEventGenerate	g_elc.p_api->softwareEventGenerate(event_num) Generate a software event in the Event Link Controller.
linkSet	g_elc.p_api->linkSet(peripheral, signal) Create a single event link.
linkBreak	g_elc.p_api->linkBreak(peripheral) Break an event link.
enable	g_elc.p_api->enable() Enable the operation of the Event Link Controller.
disable	g_elc.p_api->disable() Disable the operation of the Event Link Controller.
versionGet	g_elc.p_api->versionGet(&version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully completed.
SSP_ERR_ASSERTION	p_version is NULL.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.18.3 ELC HAL Module Operational Overview

The ELC HAL module allows the developer to create events that link various peripheral operations by using events generated by one Synergy MCU peripheral to trigger the start of operation of another Synergy MCU peripheral. The ELC HAL module APIs make it easy to create a link between two blocks (for example, from a timer to an ADC to control a periodic scan interval). By connecting various peripherals in this way, intelligent functions can be constructed that require little, if any, CPU intervention.

The following figure shows a simplified block diagram of the ELC, showing the input event sources and the peripherals that can be triggered by these events. The number of input and output triggers is specific to the S7G2 MCU. Other Synergy devices support a different number of events.

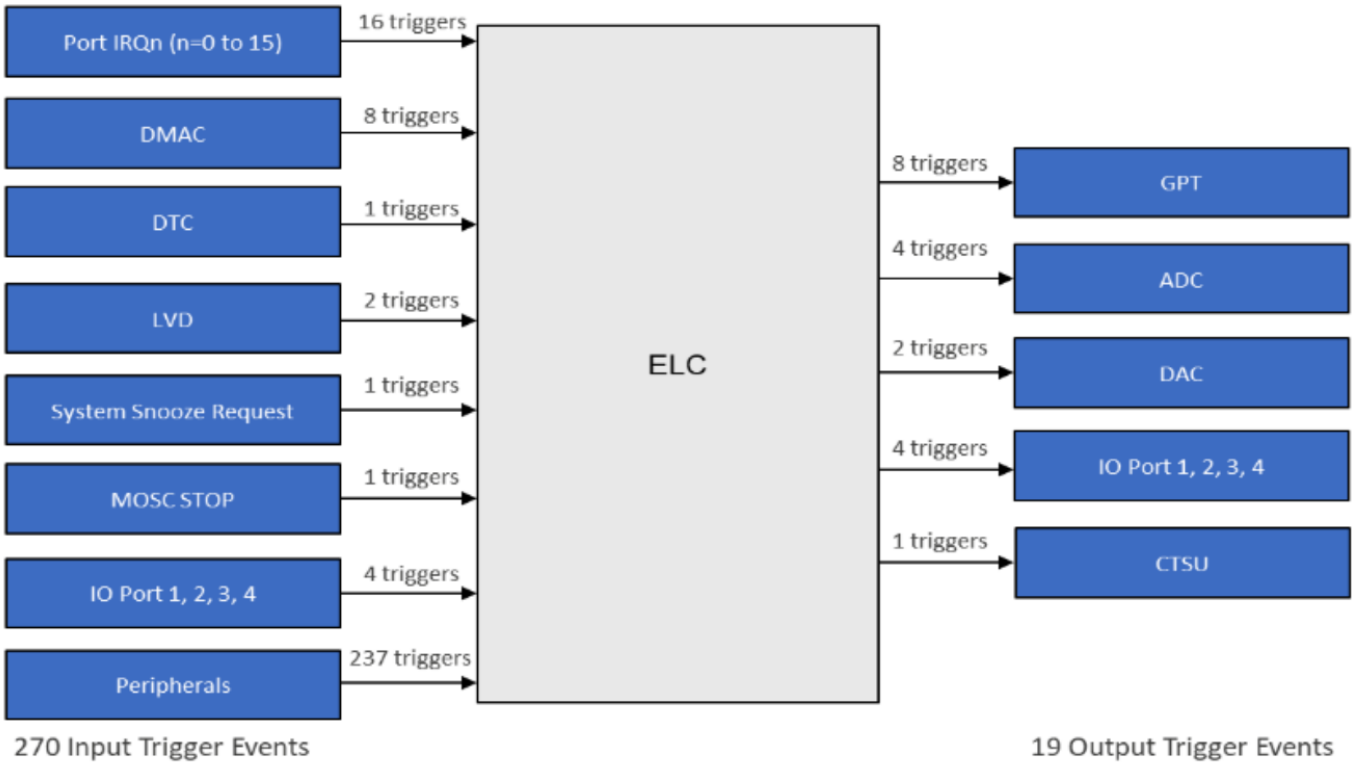


Figure 293: ELC HAL Module Trigger Events

It is possible to find the mapping of ELC peripherals in the file, bsp_elc.h, in the Synergy-generated code. The available peripherals and associated events for the S7G2 MCU are shown in the below figure. Additional information on ELC operation is also available in the associated MCU user's manual.

```
/** Possible peripherals to be linked to event signals */
typedef enum e_elc_peripheral
{
    ELC_PERIPHERAL_GPT_A                                    =    (0),
    ELC_PERIPHERAL_GPT_B                                    =    (1),
    ELC_PERIPHERAL_GPT_C                                    =    (2),
    ELC_PERIPHERAL_GPT_D                                    =    (3),
    ELC_PERIPHERAL_GPT_E                                    =    (4),
    ELC_PERIPHERAL_GPT_F                                    =    (5),
    ELC_PERIPHERAL_GPT_G                                    =    (6),
    ELC_PERIPHERAL_GPT_H                                    =    (7),
    ELC_PERIPHERAL_ADC0                                     =    (8),
    ELC_PERIPHERAL_ADC0_B                                   =    (9),
    ELC_PERIPHERAL_ADC1                                     =   (10),
    ELC_PERIPHERAL_ADC1_B                                   =   (11),
    ELC_PERIPHERAL_DAC0                                     =   (12),
    ELC_PERIPHERAL_DAC1                                     =   (13),
    ELC_PERIPHERAL_IOPORT1                                  =   (14),
    ELC_PERIPHERAL_IOPORT2                                  =   (15),
    ELC_PERIPHERAL_IOPORT3                                  =   (16),
    ELC_PERIPHERAL_IOPORT4                                  =   (17),
    ELC_PERIPHERAL_CTSU                                     =   (18),
} elc_peripheral_t;


/** Sources of event signals to be linked to other peripherals or the CPU1
 * @note This list may change based on device. This list is for S7G2.
 * */
typedef enum e_elc_event
{
    ELC_EVENT_ICU_IRQ0                                      =    (1),
    ELC_EVENT_ICU_IRQ1                                      =    (2),
    ELC_EVENT_ICU_IRQ2                                      =    (3),
    ELC_EVENT_ICU_IRQ3                                      =    (4),
.
.
.
    ELC_EVENT_ICU_IRQ12                                     =   (13),
    ELC_EVENT_ICU_IRQ13                                     =   (14),
    ELC_EVENT_ICU_IRQ14                                     =   (15),
    ELC_EVENT_ICU_IRQ15                                     =   (16),
.
.
.
    ELC_EVENT_GLCDC_LINE_DETECT                             =  (506),
    ELC_EVENT_GLCDC_UNDERFLOW_1                             =  (507),
    ELC_EVENT_GLCDC_UNDERFLOW_2                             =  (508),
    ELC_EVENT_DRW_INT                                       =  (509),
    ELC_EVENT_JPEG_JEDI                                     =  (510),
    ELC_EVENT_JPEG_JDTI                                     =  (511),
} elc_event_t;
```

Figure 294: ELC HAL Peripheral Map

ELC HAL Module Important Operational Notes and Limitations

ELC HAL Module Operational Notes

The ELC HAL module needs no pin, clocking, or interrupt configuration. It is just a 'connect' mechanism between peripherals. However, if linking I/O Ports via the ELC, the I/O pins need to be configured as inputs or outputs.

ELC HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.18.4 Including the ELC HAL Module in an Application

This section describes how to include the ELC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ELC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the ELC Driver is g_elc0. This name can be changed in the associated Properties window.)

ELC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_elc ELC Driver on r_elc	Threads	**New Stack> Driver> System> ELC Driver on r_elc**

When the ELC Driver on r_elc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

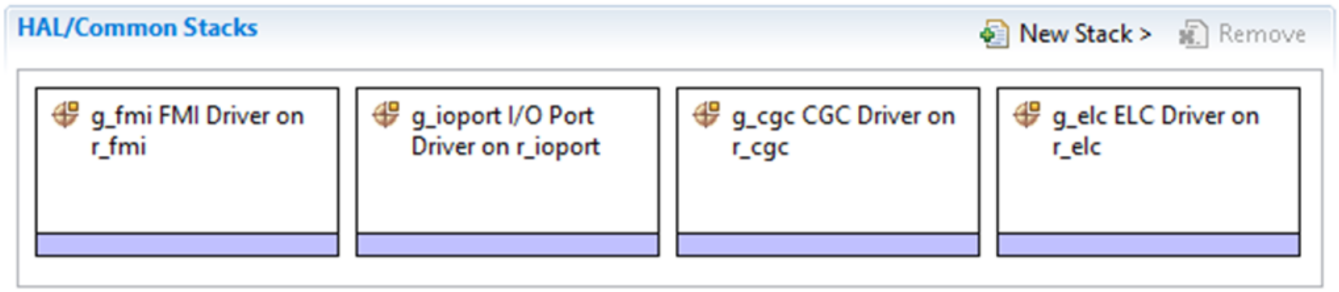


Figure 295: ELC HAL Module Stack

4.2.18.5 Configuring the ELC HAL Module

The ELC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ELC HAL Module on r_elc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_elc	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

ELC HAL Module Clock Configuration

The ELC HAL module does not require a specific clock configuration.

ELC HAL Module Pin Configuration

There are no pins associated directly with the ELC HAL Module that require configuration.

4.2.18.6 Using the ELC HAL Module in an Application

The typical steps in using the ELC HAL module in an application are:

1. Initialize the ELC using the elc_api_t::init and enable APIs (automatically done by ISDE).
2. Link a peripheral with an event using elc_api_t::linkSet API.
3. Enable the linkage with the elc_api_t::enable API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

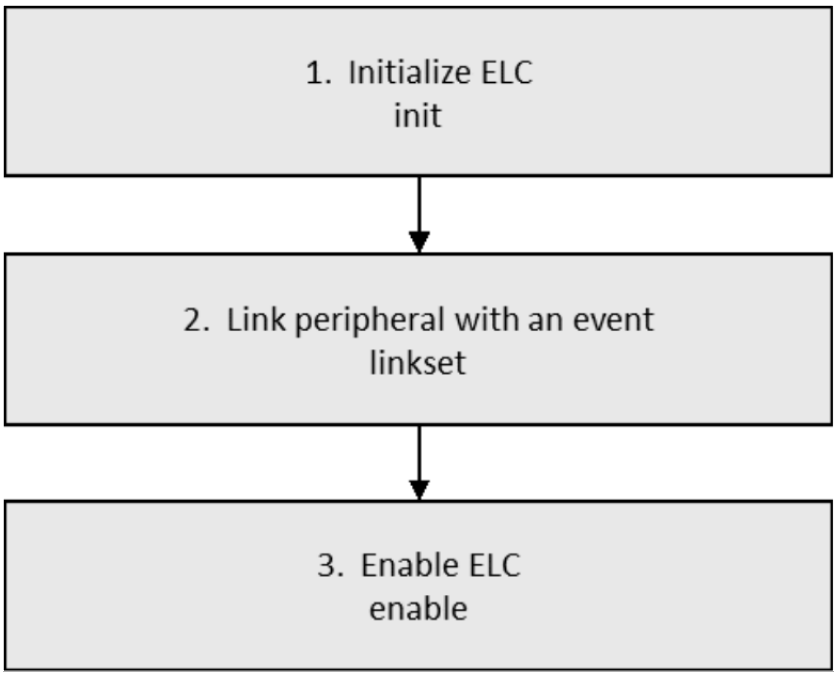


Figure 296: Flow Diagram of a Typical ELC HAL Module Application

4.2.19 External IRQ Driver

4.2.19.1 External IRQ HAL Module Introduction

The External IRQ HAL module provides an API for configuring and using external IRQ pins on Synergy MCUs. The External IRQ HAL module uses the Interrupt Controller Unit (ICU) of the Synergy MCU.

External IRQ HAL Module Features

- Supports the external interrupt pins available on the target Synergy MCU
- Supports multiple function options:
  - Enabling and disabling generation of an interrupt
  - Enabling and disabling the IRQ noise filter
  - Setting external pin IRQ trigger (Rising edge, falling edge or low level on the IRQ pin)
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

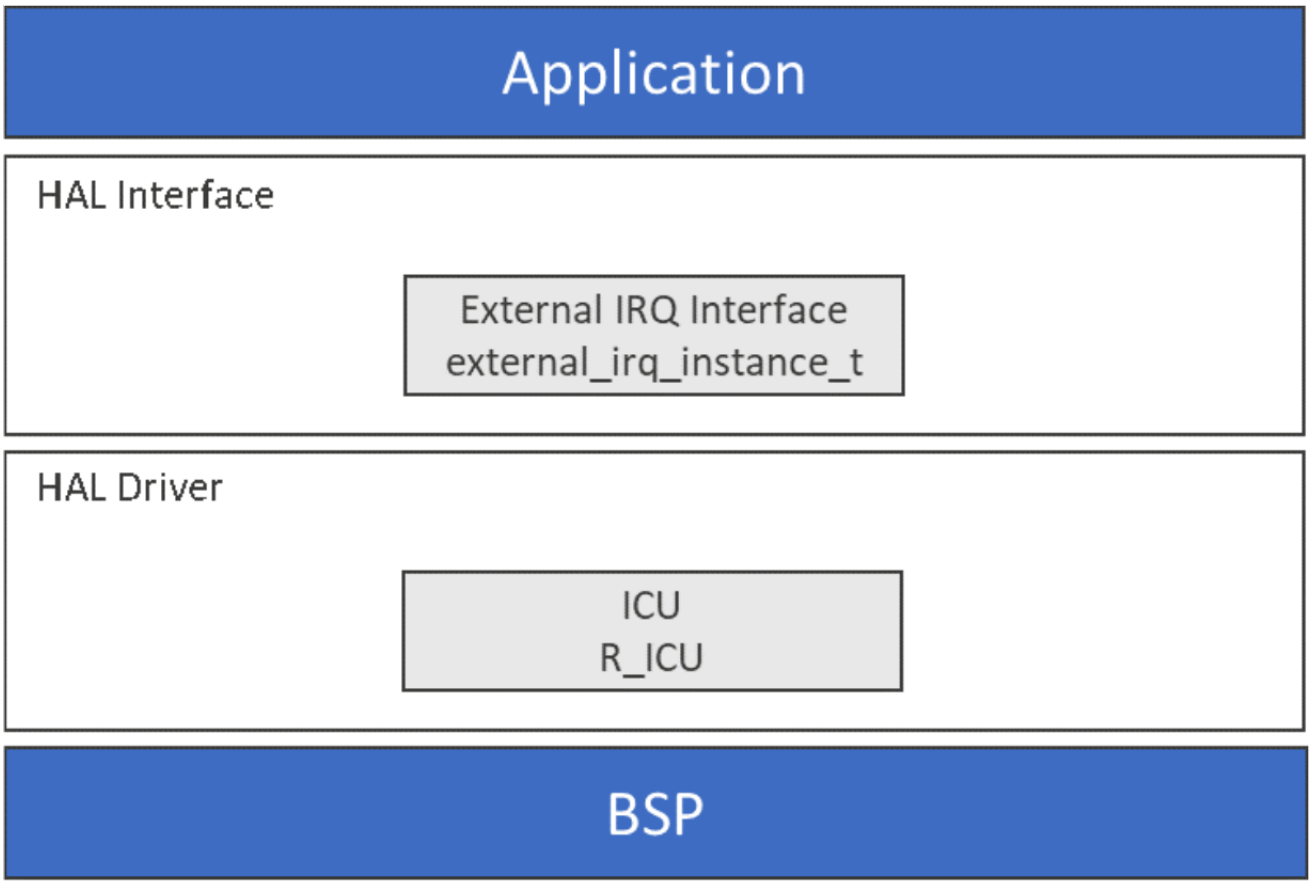


Figure 297: External IRQ HAL Module Block Diagram

The following hardware features are, or are not, supported by SSP for ICU.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Peripheral function interrupts	External pin interrupts	DTC and DMAC control	Interrupt sources for NVIC	Non-maskable interrupts (see notes)	Return from low power-mode (see notes)
S124	✓	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓

S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

Notes: The ICU module in SSP (r_icu) handles only external pin interrupts and not the other features above. Peripheral function interrupts are controlled by BSPs for each MCU and each peripheral driver modules in SSP. DTC or DMA control is handled by DTC or DMAC module in SSP (r_dtc or r_dmac). Non-maskable interrupts supported in SSP are IWDT Underflow, WDT Underflow and Voltage Monitor Interrupts. Those NMIs are controlled by IWDT, WDT or LVD modules (r_iwdt, r_wdt, or r_lvd), respectively. LVD module (r_lvd) supports the Wake Up Interrupt Enable setting. For low power mode details, see the LPM section.

MCU Group	RPEST	RECCST	BUSSST	BUSMST	SPEST
S124	☒	N/A	N/A	N/A	N/A
S128	☒	N/A	N/A	N/A	N/A
S1JA	☒	☒	☒	☒	☒
S3A1	☒	☒	☒	☒	☒
S3A3	☒	☒	☒	☒	☒
S3A6	☒	☒	☒	☒	☒
S3A7	☒	☒	☒	☒	☒
S5D3	☒	☒	☒	☒	☒
S5D5	☒	☒	☒	☒	☒
S5D9	☒	☒	☒	☒	☒
S7G2	☒	☒	☒	☒	☒

4.2.19.2 External IRQ HAL Module APIs Overview

The External IRQ HAL module defines APIs for opening, closing, and waiting for interrupt events from external pins. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

External IRQ HAL Module API Summary

Function Name	Example API Call and Description
open	g_external_irq.p_api->open(g_external_irq.p_ctrl, g_external_irq.p_cfg) Open instance and initialize.

enable	g_external_irq.p_api->enable(g_external_irq.p_ctrl) Enable callback when IRQ occurs.
disable	g_external_irq.p_api->disable(g_external_irq.p_ctrl) Disable callback when IRQ occurs.
triggerSet	g_external_irq.p_api->triggerSet(g_external_irq.p_ctrl, trigger) Set trigger.
filterEnable	g_external_irq.p_api->filterEnable(g_external_irq.p_ctrl) Enable noise filter.
filterDisable	g_external_irq.p_api->filterDisable(g_external_irq.p_ctrl) Disable noise filter.
close	g_external_irq.p_api->close(g_external_irq.p_ctrl); Close instance.
versionGet	g_external_irq.p_api->wait(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Callback is not NULL but ISR is not enabled.
SSP_ERR_IN_USE	Device in use.
SSP_ERR_NOT_OPEN	Device unopened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.19.3 External IRQ HAL Module Operational Overview

The External IRQ HAL module provides a set of API functions for controlling external interrupts. Interrupts can be triggered on rising edge, falling edge, both edges or low level of the input signal on the external IRQ pin. A digital-filtering function can be enabled to eliminate some noise on the input signal. A user-callback function is supported and is triggered each time an IRQ event occurs.

To trigger a transfer of data using the DMAC or DTC peripheral when the configured external IRQ

event occurs, configure the DMAC or DTC transfer with the activation source set to ELC_EVENT_PORTn_IRQ (where n is the IRQ channel number.)

Other peripherals can be triggered to start from an external interrupt using the Event Link Controller (ELC.) Refer to the SSP User Manual User Guide for the ELC HAL module for more information.

External IRQ HAL Module Important Operational Notes and Limitations

External IRQ HAL Module Operational Notes

- Refer to the datasheet for the target Synergy device to find the port pins which support the external interrupt functions and to obtain the external IRQ number for a given port pin.
- The external IRQ number corresponds to the channel setting in the ISDE Properties window for the External IRQ HAL module.
- The PORTn (where n is the IRQ number) interrupt must be enabled in the BSP to notify the module that the anticipated hardware event has occurred.
- A user-callback function can be registered in the external_irq_api_t::open API. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers.

    - *Note*

      Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can adversely affect the responsiveness of the system.

External IRQ HAL Module Limitations

- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.19.4 Including the External IRQ HAL Module in an Application

This section describes how to include the External IRQ HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the External IRQ Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the External IRQ Driver is g_icu0. This name can be changed in the associated Properties window.)

External IRQ HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_icu0 External IRQ Driver on r_icu	Threads	**New Stack> Driver> Input> External IRQ Driver on r_icu**

When the External IRQ Driver on r_icu is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need

only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

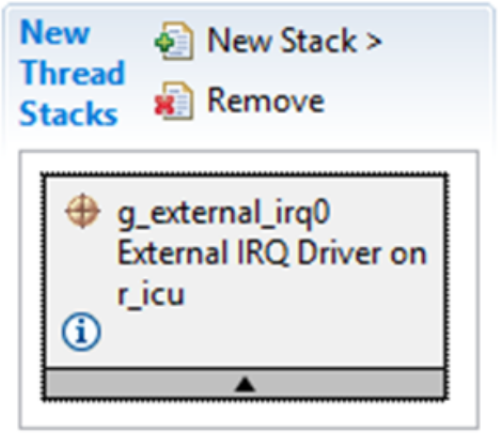


Figure 298: External IRQ HAL Module Stack

4.2.19.5 Configuring the External IRQ HAL Module

The External IRQ HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.

Trigger	Falling, Rising, Both Edges, Low Level Default: Rising	Selection for trigger event mode
Digital Filtering	Enabled, Disabled Default: Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64 Default: PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	True, False Default: True	Determines if the interrupt is enabled immediately after initialization.
Callback	NULL	A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Pin Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

External IRQ HAL Module Clock Configuration

The IRQ peripheral module does not require any specific clock settings.

External IRQ HAL Module Pin Configuration

The External IRQ peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the IRQ pins.

Pin Selection for the External IRQ HAL Module on r_icu

Resource	ISDE Tab	Pin selection Sequence
IRQ	Pins	Select **Peripherals**> **Input: IRQ**> **IRQ0**

Note

The selection sequence assumes IRQ0 is the desired hardware target for the driver.

Pin Configuration Settings for the External IRQ HAL Module on r_icu

Property	Value	Description
Operation Mode	Disabled, Enabled Default: Disabled	Select Enabled to enable interrupts.
NMI	None, P200 Default: None	Non-maskable interrupt Pin.
IRQ00:14	None, Pnn, Pmm Default: None	Interrupt request Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.19.6 Using the External IRQ HAL Module in an Application

The typical steps in using the External IRQ HAL module in an application are:

1. Initialize the External IRQ HAL module using the external_irq_api_t::open API.
2. Enable the IRQ (if needed) with the external_irq_api_t::enable API.
3. Enable the noise filter (if needed) with the external_irq_api_t::filterEnable API.
4. Change the trigger condition (only if the module is closed previously to avoid any false events) with the external_irq_api_t::triggerSet API.
5. Disable the noise filter (if enabled) with external_irq_api_t::filterDisable API.
6. Close the module (if needed) with the external_irq_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

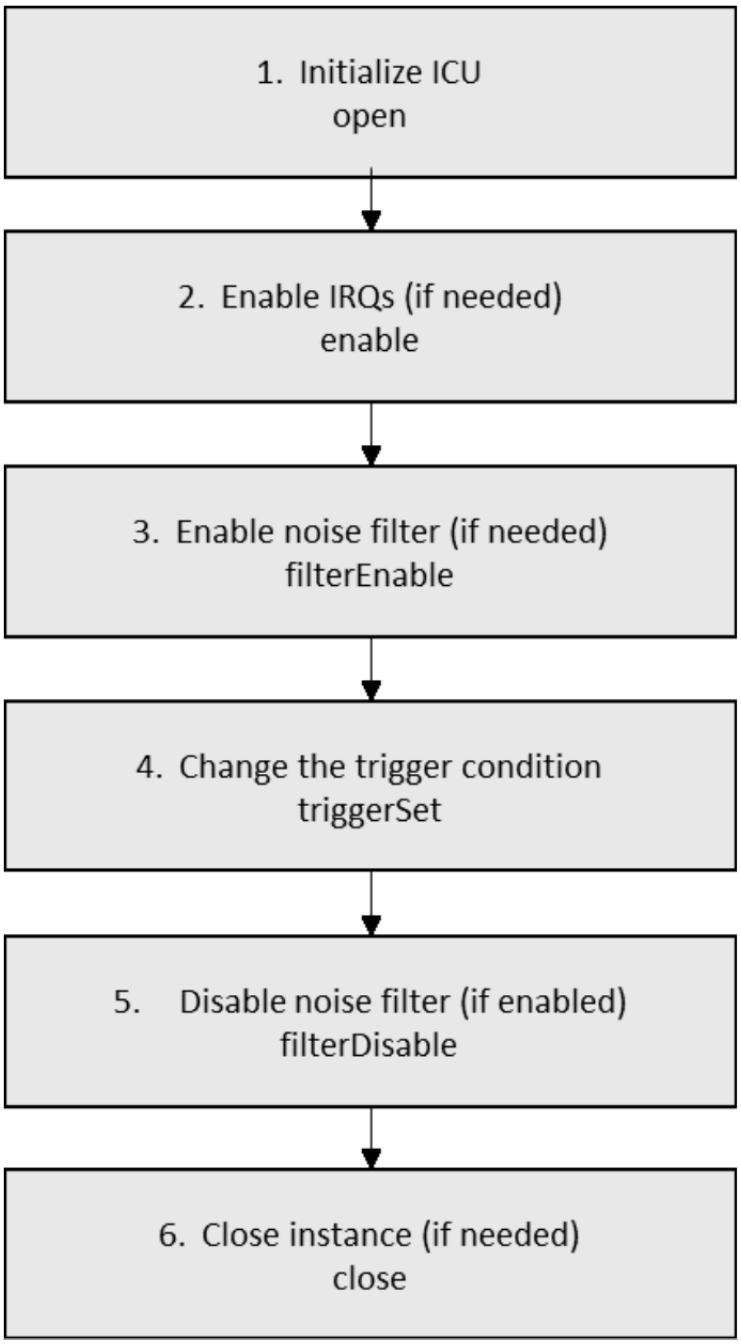


Figure 299: Flow Diagram of a Typical External IRQ HAL Module Application

4.2.20 Flash Driver

4.2.20.1 Flash HAL Module Introduction

There are two separate Flash modules: the r_flash_lp and the r_flash_hp. These modules implement a high-level API for flash memory programming applications. The High-Performance Flash module (Flash_HP) is used for programming the S7 and S5 family of MCUs. The Low-Power Flash module

(Flash_LP) is used for programming the S3 and S1 family of MCUs. The two are not interchangeable, although the APIs and other features of the modules are very similar.

Flash HAL Module Features

The Flash HAL modules APIs allow an application to read, write and erase both the data and ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts, but the API functions apply to all devices. Key features of the Flash HAL modules include:

- Support for both blocking and non-blocking erasing, reading, writing and blank-checking of data flash.
- Support for blocking erasing, reading, writing and blank checking of code flash.
- Support for callback functions for completion of non-blocking data-flash operations.
- Support for access window (write protection) for ROM Flash, allowing only specified areas of code flash to be erased or written.
- Support for boot block-swapping which allows safe rewriting of the startup program without first erasing it.

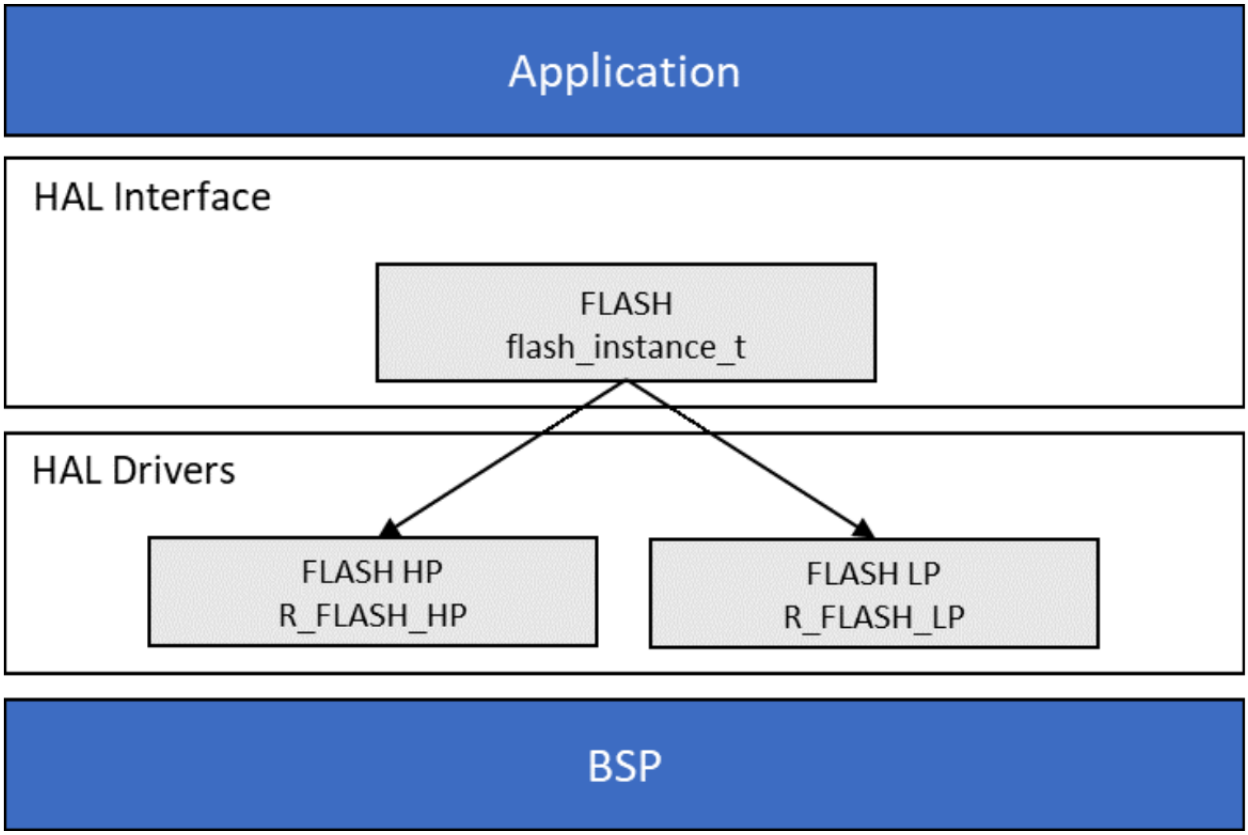


Figure 300: Flash HAL Module Block Diagram

Flash HAL HP Hardware support details

The following hardware features are, or are not, supported by SSP for the Flash_HP.

Legend:

Symbol	Meaning

✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Programming by dedicated flash memory programmer through a serial interface (serial programming)	Programming of flash memory by user program (self-programming)	Background operations (BGOs)
S124	N/A	N/A	N/A
S128	N/A	N/A	N/A
S1JA	N/A	N/A	N/A
S3A1	N/A	N/A	N/A
S3A3	N/A	N/A	N/A
S3A6	N/A	N/A	N/A
S3A7	N/A	N/A	N/A
S5D3	✓	✓	✓
S5D5	✓	✓	✓
S5D9	✓	✓	✓
S7G2	✓	✓	✓

Flash HAL LP Hardware support details

The following hardware features are, or are not, supported by SSP for the Flash_LP.

Legend:

Symbol		Meaning	
✓		Available (Tested)	
☒		Not Available (Not tested/not functional or both)	
N/A		Not supported by MCU	
MCU Group	Programming by dedicated flash memory programmer through a serial interface (serial programming)	Programming of flash memory by user program (self-programming)	Background operations (BGOs)
S124	✓	✓	✓
S128	✓	✓	✓

S1JA	✓	✓	✓
S3A1	✓	✓	✓
S3A3	✓	✓	✓
S3A6	✓	✓	✓
S3A7	✓	✓	✓
S5D3	N/A	N/A	N/A
S5D5	N/A	N/A	N/A
S5D9	N/A	N/A	N/A
S7G2	N/A	N/A	N/A

4.2.20.2 Flash HAL Module APIs Overview

The Flash HAL module defines APIs for several operations including opening, reading, erasing and closing the flash memory. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Flash HAL Module API Summary

Function Name	Example API Call and Description
open	g_flash0.p_api->open(g_flash0.p_ctrl, g_flash0.p_cfg); Open FLASH device.
write	g_flash0.p_api->write(g_flash0.p_ctrl,(uint32_t) write_buffer, FLASH_CF_32KB_BLOCK55, CODE_BLOCK_SIZE_32KB); Write FLASH device.
read	g_flash0.p_api->read(g_flash0.p_ctrl, read_buffer, DATA_FLASH_ADDR, num_bytes); Read FLASH device.
erase	g_flash0.p_api->erase(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK55,num_sectors); Erase FLASH device.
blankCheck	g_flash0.p_api->blankCheck(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK55, FLASH_DATA_BLOCK_SIZE, &blankCheck); Blank check FLASH device.
close	g_flash0.p_api->close(g_flash0.p_ctrl); Close FLASH device.
statusGet	g_flash0.p_api->statusGet(g_flash0.p_ctrl); Get Status for FLASH device.

accessWindowSet	g_flash0.p_api->accessWindowSet(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK1, FLASH_CF_32KB_BLOCK3); Set Access Window for FLASH device.
accessWindowClear	g_flash0.p_api->accessWindowClear(g_flash0.p_ctrl); Clear any existing Code Flashcode-flash access window for FLASH device.
idCodeSet	g_flash0.p_api->idCodeSet(g_flash0.p_ctrl, id_bytes, mode) Write the ID code provided to the id code registers.
reset	g_flash0.p_api->reset(g_flash0.p_ctrl); Reset function for FLASH device.
updateFlashClockFreq	g_flash0.p_api-> updateFlashClockFreq (g_flash0.p_ctrl); Update Flash clock frequency (FCLK) and recalculate timeout values.
startupAreaSelect	g_flash0.p_api->startupAreaSelect(g_flash0.p_ctrl, FLASH_STARTUP_AREA_BLOCK1); Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block. Refer to the following table for all the possible values for parameter2.
versionGet	g_flash0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

startupAreaSelect parameter2 options

Swap Type	Is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK0	False	On next reset, Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK0	False	On next reset, Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK1	False	On next reset, Startup area will be Block 1.
FLASH_STARTUP_AREA_BLOCK1	True	Startup area is immediately, but temporarily switched to Block 1.

FLASH_STARTUP_AREA_BTFLG	True	Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_IN_USE	Device in use error.
SSP_FLASH_ERR_FAILURE	Flash failure error.
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_INVALID_SIZE	Invalid size error.
SSP_ERR_INVALID_ADDRESS	Invalid address error.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified.
SSP_ERR_INVALID_ARGUMENT	Invalid argument error.
SSP_ERR_HW_LOCKED	Peripheral already in use.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_NOT_OPEN	Flash has not yet been opened.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO (background mode operation) but the Flash interrupt is not enabled.
SSP_ERR_WRITE_FAILED	Write operation failed. This may be returned if the requested Flash area is not blank.
SSP_ERR_PE_FAILURE	Failed to enter P/E mode
false	Supplied address is valid flash address on this MCU.
true	Supplied address is valid and p_block info contains the details on this address's block.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.20.3 Flash HAL Module Operational Overview

The Flash API makes the process of programming and erasing on-chip flash areas easy. Both code (user ROM) and data-flash areas are supported. The API, in its simplest form, can be used to perform

blocking erase and program operations. The term "blocking" means that when a program or erase function is called, the function does not return until the operation has finished. This API supports blocking for both code and data-flash, with non-blocking operation (also known as BGO or Background Operation) available for data-flash operations only. When a code-flash operation is on-going, you cannot access that code-flash area. If you attempt to access the code-flash area while a code-flash operation is in progress, the flash-control unit will transition into an error state.

It is important to keep in mind that even though a code-flash operation is blocking, there are several situations where the code-flash could still end up being accessed while the operation is blocking and these must be prevented. This includes:

- Vector table access if the Vector table is located in ROM.
- ROM access by an interrupt vectoring to a ROM address, even if the vector table itself is not in ROM.

A multithreaded application where multiple threads are allowed to continue to run while a code-flash operation is blocking.

Flash HAL Module Important Operational Notes and Limitations

Flash HAL Module Operational Notes

Data-Flash BGO Precautions

When using the data-flash BGO, the user ROM, RAM and external memory can still be accessed. You must ensure that the data-flash is not accessed during a data-flash operation. This includes interrupts that may access the data-flash.

Code-Flash Precautions

BGO mode is not supported for code-flash, so a code-flash operation will not return before the operation has completed. By default, the vector table resides in the user ROM (code-flash.) If an interrupt occurs during the ROM operation, then ROM will be accessed to fetch the interrupt's starting address and an error will occur.

The simplest work-around is to disable interrupts during code-flash operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must insure that if in a multithreaded environment, threads running from ROM cannot become active while a code-flash operation is in progress.

Blank Checking

The flash_api_t::blankCheck API function checks whether code or data-flash contents are blank. Note that it is not possible to write to flash (code or data) without first erasing it. The flash_api_t::blankCheck function determines whether a specified area is blank and therefore writable. In almost all cases, it is not sufficient to compare flash contents to 0xFF to determine whether the area is blank. The one exception is Flash HP code-flash. A 0xFF in Flash_HP code-flash does indicate blank. Renesas strongly recommends using the flash_api_t::blankCheck API function in all cases.

Flash Status

The flash_api_t::statusGet API function allows the application to query the 'Ready' status of the flash. This is useful in data-flash BGO operations when you choose not to use a callback function, so there is no asynchronous notification of a completed data-flash operation. In this case, the data-flash is

configured to operate in BGO mode, so once the operation is started (an erase, for example), the call returns immediately with the operation executing in the background. By calling the flash_api_t::statusGet API function, you can determine when the operation has safely completed or generated an error, and it is now safe to proceed with another flash operation.

Swap Blocks

The flash_api_t::startupAreaSelect API function allows the user to select which block - default (Block 0) or alternate (Block 1) - is used as the startup-area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary) or permanent subsequent to the next reset.

Doing a temporary switch might appear to have limited usefulness; however, if there is an access window in place such that Block 0 is write-protected, then you could do a temporary switch, update the block, and switch them back without having to touch the access window.

Flash Clock (FCLK)

The FCLK is the clock used by the Flash peripheral in performing all Flash operations. It must be >= 4 MHz for successful flash operations. As part of the flash_api_t::open function the Flash clock is checked and if < 4 MHz flash_api_t::open will return SSP_ERR_FCLK. Once the Flash API has been opened, if the FCLK frequency is changed, the flash_api_t::updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

Interrupts

Enable the flash ready interrupt only if you plan to use the data-flash BGO. In this mode, the application can initiate a data-flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (that is, flash_api_t::FLASH_EVENT_ERASE_COMPLETE)

When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the flash_api_t::open API.

Note

The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).

Access Window

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries with a starting and ending address being provided to flash_api_t::accessWindowSet. The block containing the start address is the first block. The block containing the end address is the last block. The access window then becomes the first block – last block inclusive. Anything outside this range is write protected. Invalid address information provided to flash_api_t::accessWindowSet will return SSP_ERR_INVALID_ADDRESS. An access window may be removed by calling the flash_api_t::accessWindowClear API function

ID code set

Allows user to program the ID bytes rather than having the pre-programmed ID bytes with S-record.

Flash HAL Module Limitations

- The High-Performance Flash module (Flash_HP) is the API used for programming the S7 and S5 family of MCUs.
- The Low-Power Flash module (Flash_LP) is the API used for programming the S3 and S1 family of MCUs.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.20.4 Including the Flash HAL Module in an Application

This section describes how to include the Flash HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Flash Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Flash Driver is g_flash0. This name can be changed in the associated Properties window.)

Flash HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_flash0 Flash Driver on r_rflash_hp	Threads	**New Stack> Driver> Storage> Flash Driver on r_flash_hp**
g_flash0 Flash Driver on r_rflash_lp	Threads	**New Stack> Driver> Storage> Flash Driver on r_flash_lp**

When the Flash Driver on r_flash_hp or r_flash_lp is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

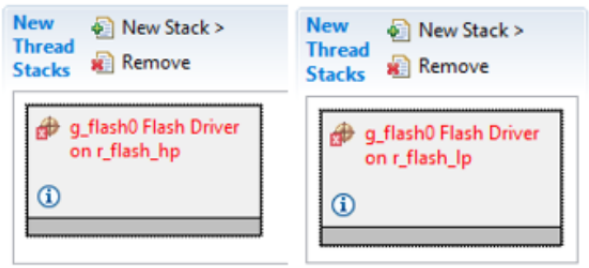


Figure 301: Flash HAL Module Stack

4.2.20.5 Configuring the Flash HAL Module

The Flash HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Flash HAL Module on r_flash_hp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Code Flash Programming Enable	Enable, Disabled Default: Disabled	Controls whether or not Code Flash programming is enabled. Disabling reduces the amount of ROM used by the API.
Name	g_flash0	Module name.
Data Flash Background Operation	Enabled, Disabled Default: Enabled	Enabling allows Flash API calls that reference Data Flash to return immediately, with the operation continuing in the background.
Callback	NULL	Callback function called when a Data Flash BGO operation completes or errors. A user callback function can be registered in open. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Flash Ready Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Flash ready interrupt priority selection.

Flash Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Flash error interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Flash HAL Module on r_flash_lp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Code Flash Programming Enable	Enable, Disabled Default: Disabled	Controls whether or not Code Flash programming is enabled. Disabling reduces the amount of ROM used by the API.
Name	g_flash0	Module name.
Data Flash Background Operation	Enabled, Disabled Default: Enabled	Enabling allows Flash API calls that reference Data Flash to return immediately, with the operation continuing in the background.
Callback	NULL	Callback function called when a Data Flash BGO operation completes or errors. A user callback function can be registered in open. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Flash Ready Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Flash ready interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to disable code-flash programming to reduce the code size of the driver.

Flash HAL Module Clock Configuration

Enable the flash-ready interrupt only if you plan to use the data-flash BGO (background mode operation.) In this mode, the application can initiate a data-flash operation and then be asynchronously notified of its completion (or an error) using a user-supplied callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, FLASH_EVENT_ERASE_COMPLETE.)

To enable interrupts, set the priority of the FCU > FRDYI interrupt on the **ICU** tab of the Project Configurator in e^2 studio. This sets BSP_IRQ_CFG_FCU_FRDYI in synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the FLASH FRDYI interrupt is enabled in the BSP, the corresponding ISR will be defined in the Flash driver. The ISR will call a user-callback function if one was registered in open.

Note

The Flash HP supports an additional flash-error interrupt, and if BGO mode is enabled, then both FRDYI and FIFERR interrupts must be given a priority.

Flash HAL Module Pin Configuration

The flash circuit uses FCLK as its clock. FCLK must be <= 4 MHz. If this clock rate changes after the flash_api_t::open API is called, then you must call flash_api_t::updateFlashClockFreq API to inform the flash API of the change.

4.2.20.6 Using the Flash HAL Module in an Application

The typical steps in using the Flash HAL module in an application are:

1. Initialize the Flash HAL using the flash_api_t::open API.
2. Disable Interrupts.
3. Blank check a code flash area with flash_api_t::blankCheck API.
4. Erase one or more code-flash blocks with flash_api_t::erase API.
5. Write to code-flash with the flash_api_t::write API.
6. Enable Interrupts.
7. Blank check a data flash area with flash_api_t::blankCheck API.
8. Erase one or more data-flash blocks using the flash_api_t::erase API.
9. Write to data-flash using the flash_api_t::write API.
10. Enable Data Flash BGO mode and assign a callback function.
11. Erase one or more data-flash blocks using the flash_api_t::erase API.
12. Verify that the Erase has completed successfully by checking the event info passed to the callback.
13. Close using the flash_api_t::close API if finished with all Flash operations.

These common steps are illustrated in a typical operational flow diagram in the following figure:

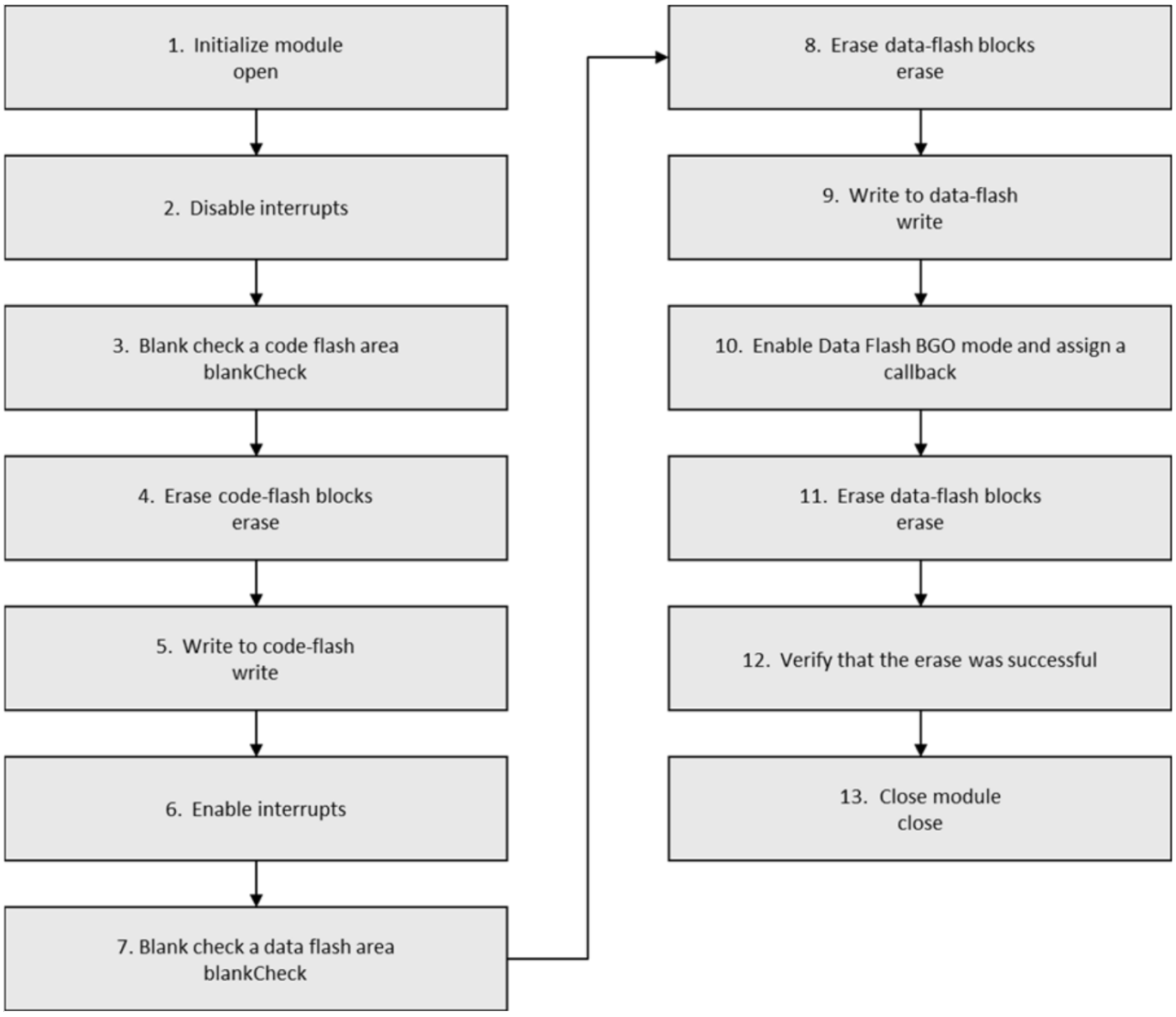


Figure 302: Flow Diagram of a Typical Flash HAL Module Application

4.2.21 FMI Driver

4.2.21.1 FMI HAL Module Introduction

The FMI HAL module provides a high-level API for applications that read records from the Factory MCU Information Flash Table and uses the Flash Interface on the Synergy MCU.

FMI HAL Module Features

The FMI HAL module reads the FMIFRT (Factory MCU Information Flash Root Table) on a Synergy microcontroller, looking up the address of the start of the table in flash. The module sets the caller's

pointer to the Product Information record from the table. This information may be used to determine the capabilities of features specific to this MCU package. Information available from the FMI HAL module includes:

- Product information (that is, product name, package, pin count and temperature range)
- Product features (version major, version minor and variant data)
- Event information such as interrupts and events

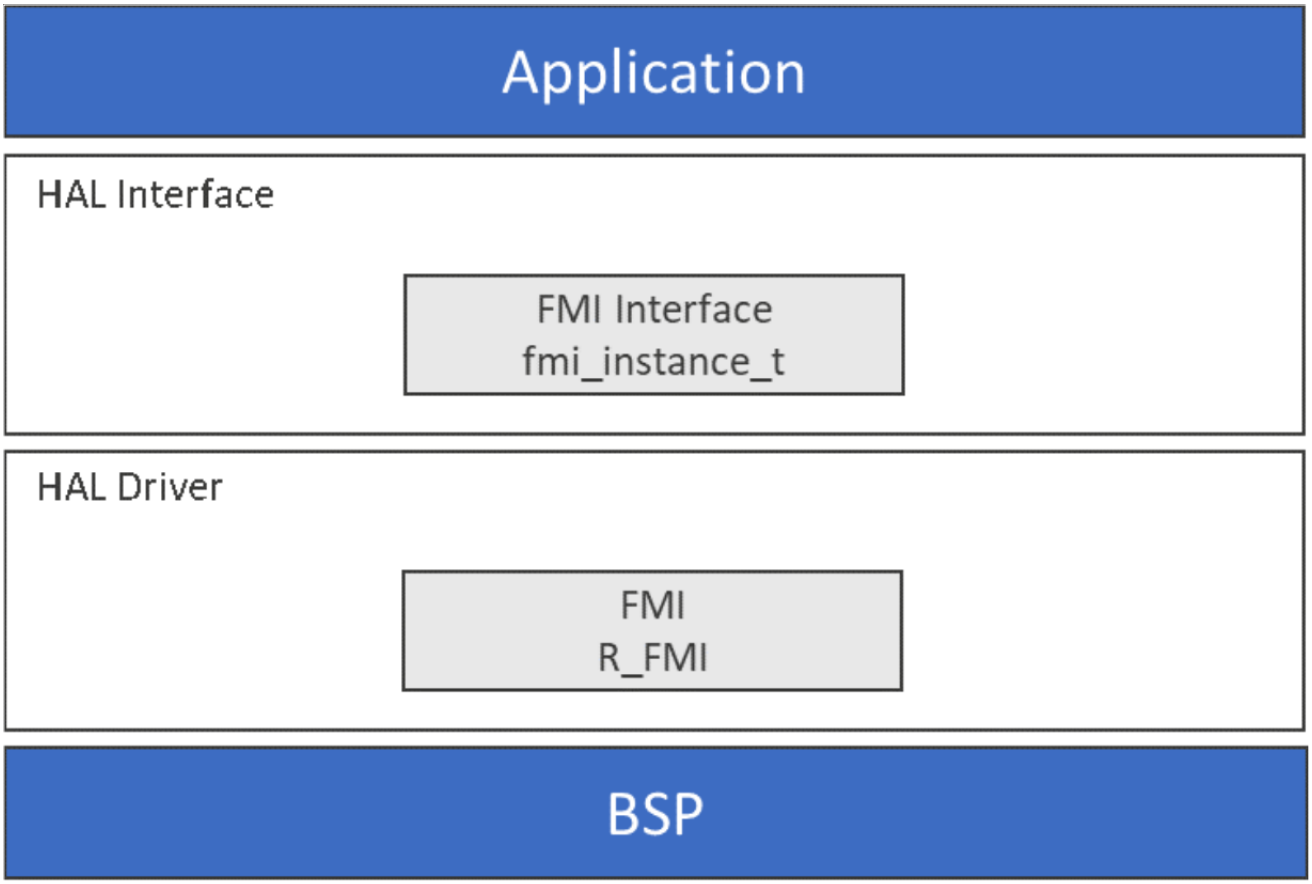


Figure 303: FMI HAL Module Block Diagram

4.2.21.2 FMI HAL Module APIs Overview

The FMI HAL module defines an API for accessing the FMIFRT. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

FMI HAL Module API Summary

Function Name	Example API Call and Description
init	g_fmi.p_api->init(); Initialize the FMI base pointer.
productInfoGet	g_fmi.p_api->productInfoGet(&g_pp_product_info); Get product information record address into g_pp_product_info pointer.

uniqueIdGet	g_fmi.p_api->uniqueIdGet(&g_p_unique_id); Copy the unique ID into the g_p_unique_id pointer.
productFeatureGet	g_fmi.p_api->productFeatureGet(&g_ssp_feature, &g_feature_info); Get feature information and store it in g_feature_info pointer.
eventInfoGet	g_fmi.p_api->eventInfoGet(&g_ssp_feature,SSP_SIGNAL_GPT_COUNTER_OVERFLOW, &g_event_info); Get event information and store it in g_event_info pointer.
versionGet	g_fmi.p_api->versionGet(&g_p_version); Get the driver version based on compile time macros.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_FMI_DATA	The FMI data table provided is not valid
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this MCU
SSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this MCU
SSP_ERR_INTERNAL	Requested feature is in a format not supported at this time
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found
SSP_ERR_ASSERTION	Caller's pointer is null
SSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not valid

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.21.3 FMI HAL Module Operational Overview

The FMI HAL module retrieves the product information record address and populates the fmi_product_info_t structure using the fmi_api_t::productInfoGet API.

The FMI HAL module copies unique ID and populates the fmi_unique_id_t structure using the fmi_api_t::uniqueIdGet API.

The FMI HAL module gets feature information and populates the fmi_feature_info_t structure using

the fmi_api_t::productFeatureGet API.

The FMI HAL module fetches event information and populates the fmi_event_info_t structure using the fmi_api_t::eventInfoGet API.

The FMI HAL module gets code version and API version in ssp_version_t structure using the fmi_api_t::versionGet API.

For details, refer the FMI HAL module source code and SSP user manual.

FMI HAL Module Important Operational Notes and Limitations

FMI HAL Module Operational Notes

- The fmi_product_info_t::unique_id is deprecated. It does not contain a unique ID if the factory MCU information is linked in by the application code. Use fmi_api_t::uniqueIdGet for the unique ID.

FMI HAL Module Limitations

- For limitations of FMI HAL Interface and its implementation, see the latest SSP release notes.
- The FMI Driver has been tested on the S7G2 (WS2) Synergy microcontroller family using the FMIFRT peripheral register. It is the only Synergy MCU that is currently programmed with data in the Factory MCU Information Table.

4.2.21.4 Including the FMI HAL Module in an Application

This section describes how to include the FMI HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the FMI Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the FMI Driver is g_fmi0. This name can be changed in the associated Properties window.)

FMI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_fmi FMI Driver on r_fmi	**Threads> HAL/Common**	**New Stack> Driver> System> FMI Driver on r_fmi**

When the FMI Driver on r_fmi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and

displays possible choices.

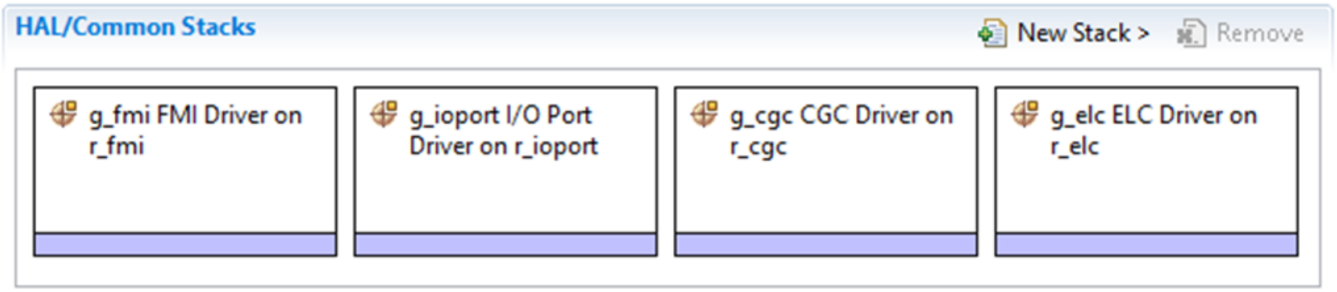


Figure 304: FMI HAL Module Stack

4.2.21.5 Configuring the FMI HAL Module

The FMI HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the FMI HAL Module on r_fmi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
SSP MCU Information Symbol Name	g_fmi_data	This symbol maps to the base address where the factory flash table information will be found. It should not be modified.

Part Number Mask	0xFE00	Each bit represents one character in the Synergy part number, where the MSB is the first character in the part number ('R'). Set bits to ensure the part number in the MCU factory flash matches the part number in the SSP MCU Information. The default mask checks everything except operating temperature, software ID, and quality ID.
Name	g_fmi	Module instance name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

FMI HAL Module Clock Configuration

No specific clock configurations are required for the FMI HAL Module.

FMI HAL Module Pin Configuration

No specific pin configurations are required for the FMI HAL Module.

4.2.21.6 Using the FMI HAL Module in an Application

The typical steps in using the FMI HAL Module in an application are:

1. Initialize the FMI using the fmi_api_t::init API, it is automatically initialized after Reset.
2. Use the fmi_api_t::productInfoGet API to get product information.
3. Use the fmi_api_t::uniqueIdGet API to get unique ID.
4. Use the fmi_api_t::productFeatureGet API to get feature information.
5. Use the fmi_api_t::eventInfoGet API to get event information.
6. Use the fmi_api_t::versionGet API to get driver version information.

These common steps are illustrated in a typical operational flow diagram in the following figure:

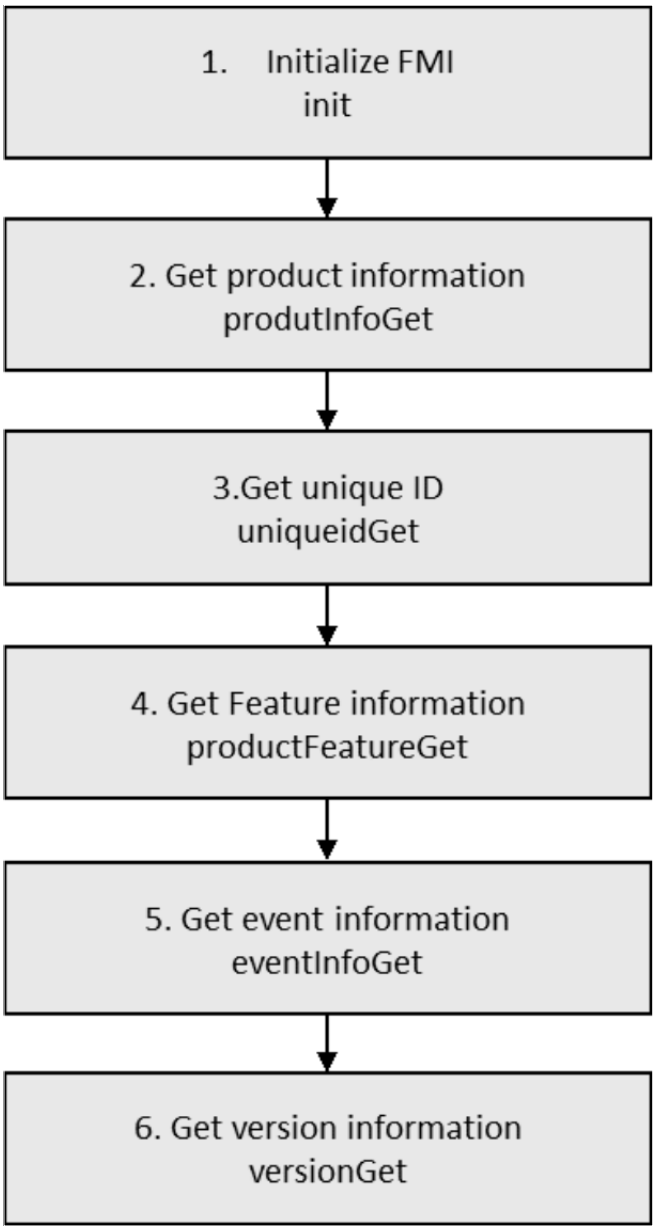


Figure 305: Flow Diagram of a Typical FMI HAL Module Application

4.2.22 Timer Driver on r_gpt

4.2.22.1 GPT HAL Module Introduction

The General PWM Timer (GPT) HAL module provides a high-level API for timer applications and uses the GPT peripheral on the Synergy MCU. A user-defined callback can be created to respond to a timer event.

GPT HAL Module Features

The GPT HAL module configures a timer to a user-specified period. When the period elapses, any of the following events can occur:

- CPU interrupt that calls a user callback function, if provided
- Toggle a port pin
- Data transfer using DMAC/DTC if configured with Transfer Interface
- Starting of another peripheral if configured with events and peripheral definitions

General PWM Timer (GPT)

- PCLKD as core clock
- Two I/O pins per channel

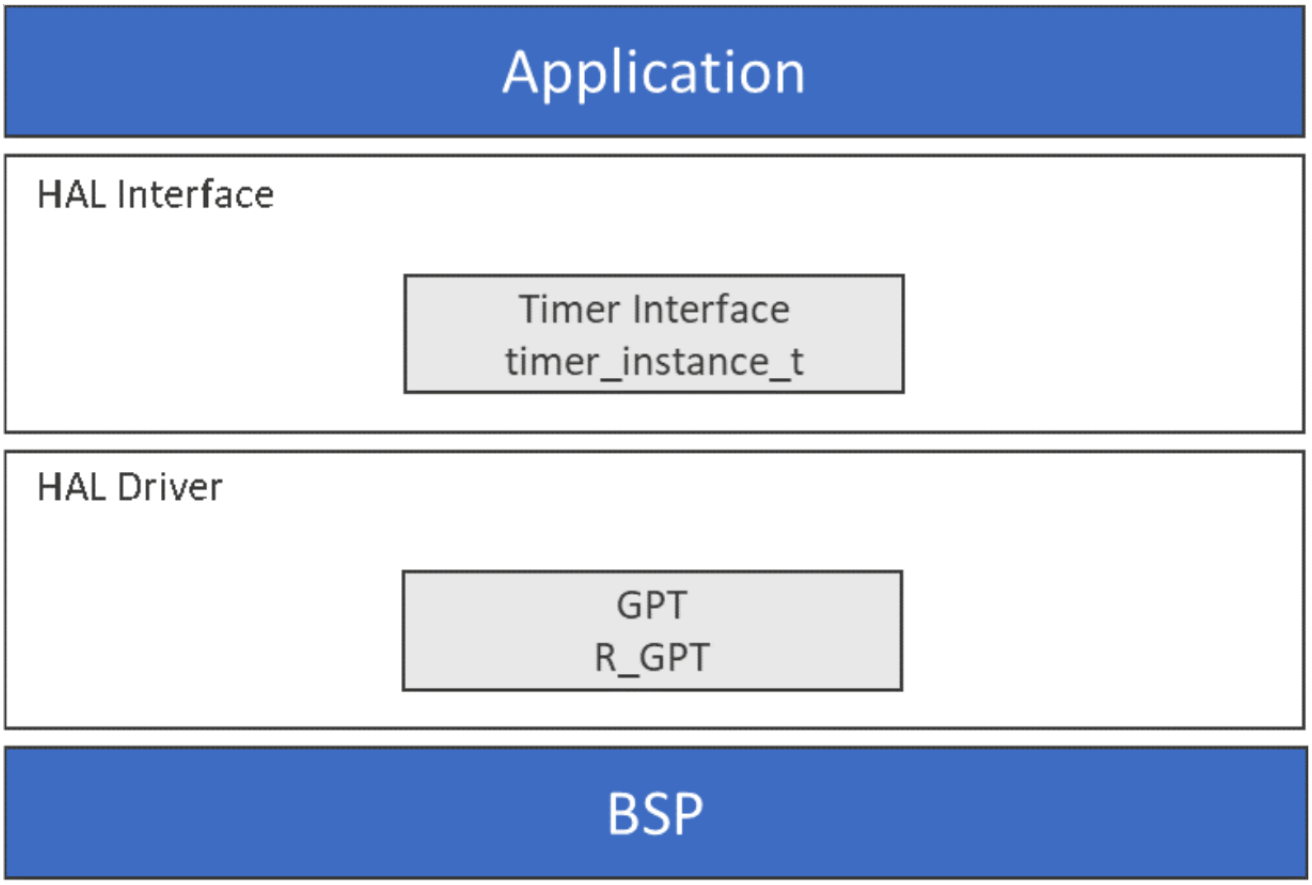


Figure 306: GPT HAL Module Block Diagram

GPT Hardware support details

The following hardware features are, or are not, supported by SSP for GPT.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A	Not supported by MCU

MCU Group	Saw Waves	Triangle Waves	PWM waveform for controlling brushless DC motors	Compare match output for Low, High, and Toggle	Input capture function	Automatic addition of dead time
S124	✓	☒	☒	✓	☒	☒
S128	✓	☒	☒	✓	☒	☒
S1JA	✓	☒	☒	✓	☒	☒
S3A1	✓	☒	☒	✓	☒	☒
S3A3	✓	☒	☒	✓	☒	☒
S3A6	✓	☒	☒	✓	☒	☒
S3A7	✓	☒	☒	✓	☒	☒
S5D3	✓	☒	☒	✓	☒	☒
S5D5	✓	☒	☒	✓	☒	☒
S5D9	✓	☒	☒	✓	☒	☒
S7G2	✓	☒	☒	✓	☒	☒

MCU Group	PWM Mode	Phase Count Function	One-Shot Operation	Event link function through ELC HAL driver	Noise filtering function
S124	✓	☒	✓	☒	☒
S128	✓	☒	✓	☒	☒
S1JA	✓	☒	✓	☒	☒
S3A1	✓	☒	✓	☒	☒
S3A3	✓	☒	✓	☒	☒
S3A6	✓	☒	✓	☒	☒
S3A7	✓	☒	✓	☒	☒
S5D3	✓	☒	✓	☒	☒
S5D5	✓	☒	✓	☒	☒
S5D9	✓	☒	✓	☒	☒
S7G2	✓	☒	✓	☒	☒

4.2.22.2 GPT HAL Module APIs Overview

The GPT HAL module defines APIs to open, start, stop, read status, trim and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

GPT HAL Module API Summary

Function Name	Example API Call and Description
open	g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg) Initial configuration.
stop	g_timer0.p_api->stop(g_timer0.p_ctrl) Stop the counter.
start	g_timer0.p_api->start(g_timer0.p_ctrl) Start the counter.
reset	g_timer0.p_api->reset(g_timer0.p_ctrl) Reset the counter initial value.
counterGet	g_timer0.p_api->counterGet(&value) Get current counter value and store it in the provided pointer, value.
periodSet	g_timer0.p_api->periodSet(g_timer0.p_ctrl, period, unit) Set the time until the timer expires.
dutyCycleSet	g_timer0.p_api->dutyCycleSet(g_timer0.p_ctrl, period, unit, pin) Sets the time until the duty cycle expires.
infoGet	g_timer0.p_api->infoGet(&info) Get the time until the timer expires in clock counts and store it in provided pointer, info.
close	g_timer0.p_api->close(g_timer0.p_ctrl) Allows driver to be reconfigured and may reduce power consumption.
versionGet	g_timer0.p_api->versionGet(g_timer0.p_ctrl, &version) Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Operation is successful.

SSP_ERR_ASSERTION	Parameter is NULL or configuration setting is not allowed.
SSP_ERR_IN_USE	The channel specified has already been opened.
SSP_ERROR_NOT_OPEN	The channel is not open.
SSP_ERR_INVALID_ARGUMENT	Invalid argument provided.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.22.3 GPT HAL Module Operational Overview

The GPT HAL module configures a timer to a user-specified period. When the period elapses, the CPU can be interrupted, a port pin can be toggled, a transfer of data using the DMAC or DTC can be initiated, or another peripheral can be triggered to begin operation.

The following figure shows a flowchart for toggling a port pin or generating a CPU interrupt after a specified period:

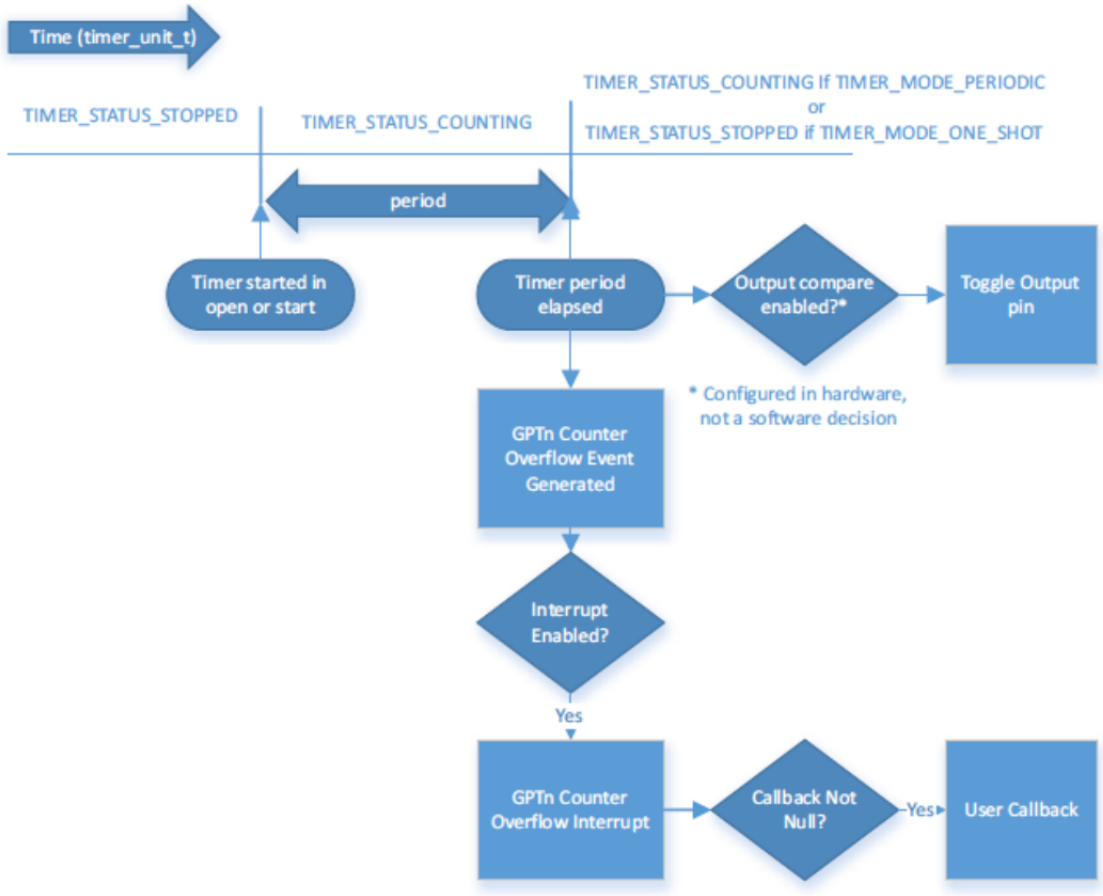


Figure 307: GPT HAL Module Timer-Periodic or One-Shot Mode

Two different timer modules, the GPT and the AGT, are supported in the SSP. The following sections

provide information on both modules so that the developer can compare and contrast the capabilities of each module for a particular application. For additional information on the AGT, refer to the AGT User's Guide.

The GPT module is recommended for most generic timer applications, but either module can be used for a basic timer functionality. The following use cases describe why one timer module would be preferred over the other.

Selecting the GPT Timer Module

The GPT module uses a high-resolution 32-bit counter that can only be clocked by PCLKD. There are more GPT channels than AGT channels on Synergy devices, so using GPT is less likely to cause a resource conflict.

Selecting the AGT Timer Module

The AGT module uses a 16-bit counter that can be clocked by PCLKB, LOCO, or Fsub. If clocked by LOCO or Fsub, the AGT interrupt can be used to wake the MCU from sleep modes. There are two channels, and channel 1 can be clocked by channel 0 underflow, effectively creating a 32-bit cascaded timer.

GPT HAL Module Important Operational Notes and Limitations

GPT HAL Module Operational Notes

The maximum time period depends on the timer type and the input clock frequency.

- On a GPT with 32-bit resolution with PCLKD running at 120 MHz, the maximum period is approximately 36650 seconds, which is just over 10 hours (GPT Count Clock is PCLKD/1024).
- On a GPT with 16-bit resolution with PCLKD running at 32 MHz, the maximum period is approximately 2.09 seconds (GPT Count Clock is PCLKD/1024).

The GPT counter overflow interrupt must be enabled in the following situations:

1. To get a software interrupt when the timer period has elapsed.
2. To use one-shot mode

When counter overflow interrupt is enabled, the corresponding ISR is linked in the vector table. The ISR calls a user callback function if one was registered in open.

Note: Interrupts may be skipped when used with the DTC peripheral with irq set to TRANSFER_IRQ_END.

GPT Output Timer Signal

If the timer output is configured (GTIOCA/B Output Enabled set to True), the output pin will start at the GTIOCA/B Stop Level and toggle every time the period elapses, beginning with the first time the period elapses after the timer is started.

In one-shot mode, the output is also configured to toggle when the timer starts counting. This generates a pulse - the timer toggles from the stop level when counting begins and toggles back to the stop level when counting ends.

Timer Period Calculation

The timer period is defined as the time until the timer expires. When output compare is used, the output pin will toggle once per period, so the traditional period (from rising edge to rising edge) is twice the period specified in the software.

Runtime period calculation based on the current clock settings is available from open and periodSet.

- If the specified Period Unit is different than the Raw Counts, the period is calculated using the current timer clock frequency. The current timer clock frequency is determined using the cgc_api_t::systemClockFreqGet API. This frequency will be used in the appropriate formula from the following table as clk_freq_hz.

Timer Period Calculation

Timer Units	Formula
TIMER_UNIT_PERIOD_NSEC	Counts = (period * clk_freq_hz) / 1000000000
TIMER_UNIT_PERIOD_USEC	Counts = (period * clk_freq_hz) / 1000000
TIMER_UNIT_PERIOD_MSEC	Counts = (period * clk_freq_hz) / 1000
TIMER_UNIT_PERIOD_SEC	Counts = (period * clk_freq_hz)
TIMER_UNIT_FREQUENCY_HZ	Counts = (clk_freq_hz) / period
TIMER_UNIT_FREQUENCY_KHZ	Counts = (clk_freq_hz) / 1000 * period

If the requested period is larger than the counter size (32-bit or 16-bit), the driver selects the smallest divisor that allows the result to fit in the counter size. If the counter value is larger than the counter size with the largest divisor (1024), an error code (SSP_ERR_INVALID_ARGUMENT) is returned.

Triggering DMAC/DTC with GPT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_GPTn_COUNTER_OVERFLOW (where n is the GPT channel number). See the DMAC or DTC guides for further information.

Note

If you use the timer in one-shot mode with the DTC, the entire transfer will complete before the interrupt stops the timer if irq is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set irq to TRANSFER_IRQ_EACH, or use the DMAC for the transfer.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals.

Free Running Counter Mode

To use the GPT as a free running counter, set the Period to 0xFFFFFFFF for a 32-bit timer or 0xFFFF for a 16-bit timer and the Period Unit to Raw Counts in the Properties window of the Synergy Configuration tool. Stop and start the timer using the timer_api_t::stop API and the timer_api_t::start API. Check the counter value using the timer_api_t::counterGet API. Reset the timer using the timer_api_t::reset API. If the counter overflows, handle the counter overflow in the callback.

GPT PWM Mode

To use the GPT in PWM mode, set the Period and Duty cycle and select the duty cycle range.

The driver provides two options to select the duty cycle range:

1. Shortest duty cycle off: In this case, the lowest duty cycle obtained will be limited to 2 raw counts. But the configuration will be limited to 1 raw counts(because the hardware will add 1 extra clock cycle in ON time, hence if configured Dutycycle is 1 raw count, the user will be getting 2 raw counts in ON time).
2. Shortest duty cycle on: In this case, the lowest duty cycle of 1 raw count can be achieved and lowest duty cycle to configure will be limited to 1 raw counts and the longest will be (Period – 2). In this case, the 1 extra clock cycle will be added by hardware in OFF time.

GPT HAL Module Limitations

- For GPT Power Down, the GPT module does not set the Module Stop bit (MSTP) for GPT in the timer_api_t::close API. This is intentional because the GPT module stop bits control the power to multiple GPT channels, and the GPT module cannot know if other GPT modules are used in the application.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.22.4 Including the GPT HAL Module in an Application

This section describes how to include the GPT HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Timer Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Timer Driver is g_timer0. This name can be changed in the associated Properties window.)

GPT HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_timer0 Timer Driver on r_gpt	**Threads> HAL/Common**	**New Stack> Driver> Timers> Timer Driver on r_gpt**

When the Timer Driver on r_gpt is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

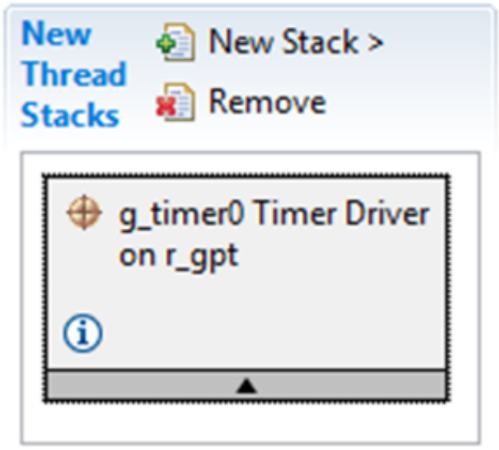


Figure 308: GPT HAL Module Stack

4.2.22.5 Configuring the GPT HAL Module

The GPT HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic, One Shot, PWM Default: Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.

Period Value	10	See Timer Period Calculation.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Milliseconds	See Timer Period Calculation.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	True, False Default: True	Set to true to start the timer after configuring or false to leave the timer stopped until timer_api_t::start is called.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	A user callback function can be registered in timer_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Overflow interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

GPT HAL Module Clock Configuration

The GPT timer is clocked based on the PCLKD frequency. You can set the PCLKD frequency using the clock configurator in the ISDE Configuring Clocks tab, or the CGC Interface at run-time.

GPT HAL Module Pin Configuration

The GPT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table lists the method used to select pins within the SSP configuration window and the subsequent table lists an example selection for the associated pins:

Pin Selection for the GPT HAL Module on r_gpt

Resource	ISDE Tab	Pin selection Sequence
GPT	Pins	Select **Peripherals**> **Timer: GPT**> **GPT0**

Note

The selection sequence assumes GPT0 is the desired hardware target for the driver.

Pin Configuration Settings for GPT HAL Driver

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only Default: Mixed	Select pin group mapping.
Operation Mode	Disabled, GTIOCA or GTIOCB, GTIOCA and GTIOCB Default: Disabled	Select timer operation mode.
GTIOCA:	None, P300, P512 Default: P512	GTIOCA Pin.
GTIOCB:	None, P108, P511 Default: P511	GTIOCB Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy

MCUs and Synergy Kits may have different available pin configuration settings.

4.2.22.6 Using the GPT HAL Module in an Application

The typical steps in using the GPT HAL module in an application are:

1. Initialize the GPT HAL module using the timer_api_t::open API.
2. Start the GPT HAL module by calling the timer_api_t::start API if the Auto Start property is False.
3. Respond to the timer callback as needed (application code).

Note

The GPT period and duty cycle parameters can be reconfigured based on the application needs using the timer_api_t::periodSet() and the timer_api_t::dutyCycleSet.

In PWM mode, there will be one extra PCLK added by the hardware in ON time if Duty cycle range is selected to GPT_SHORTEST_LEVEL_OFF and in OFF time if the Duty cycle range is selected to GPT_SHORTEST_LEVEL_ON.

These common steps are illustrated in a typical operational flow diagram in the following figure:

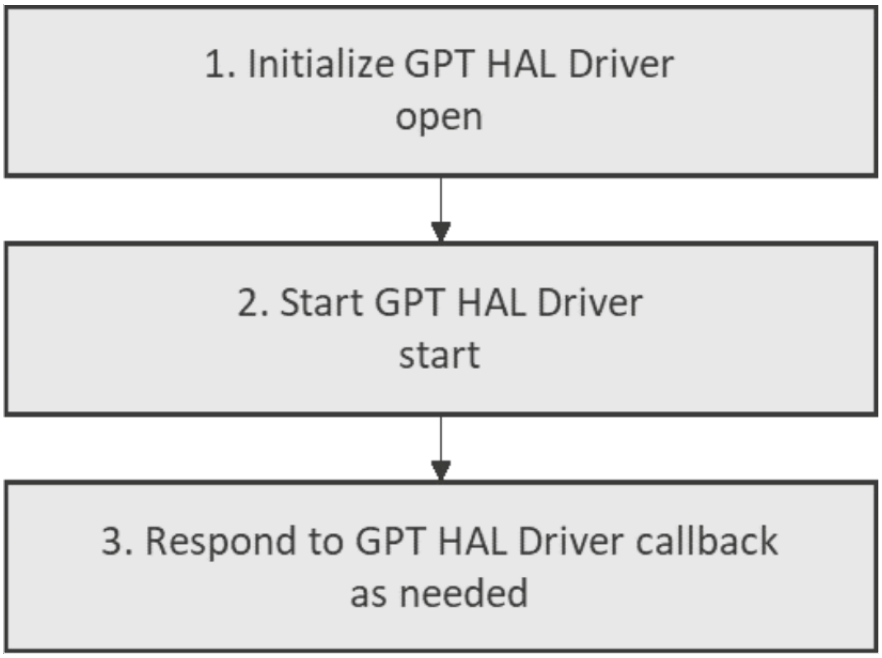


Figure 309: Flow Diagram of a Typical GPT HAL Module Application

4.2.23 I2C SCI Driver

4.2.23.1 I2C SCI HAL Module Introduction

The I2C SCI Master HAL module provides a high-level API for I2C industry standard serial device communication applications and uses the SCI peripheral on the Synergy MCU device. Callbacks are

provided for transmit complete and receive complete.

I2C SCI HAL Module Features

- Support for I2C SCI operations
- Supports following operations with a slave I2C SCI device
    - Read
    - Write
    - Reset
- Callback support
    - Transfer aborted
    - Transmit complete (number of bytes transmitted provided)
    - Receive complete (number of bytes received provided)

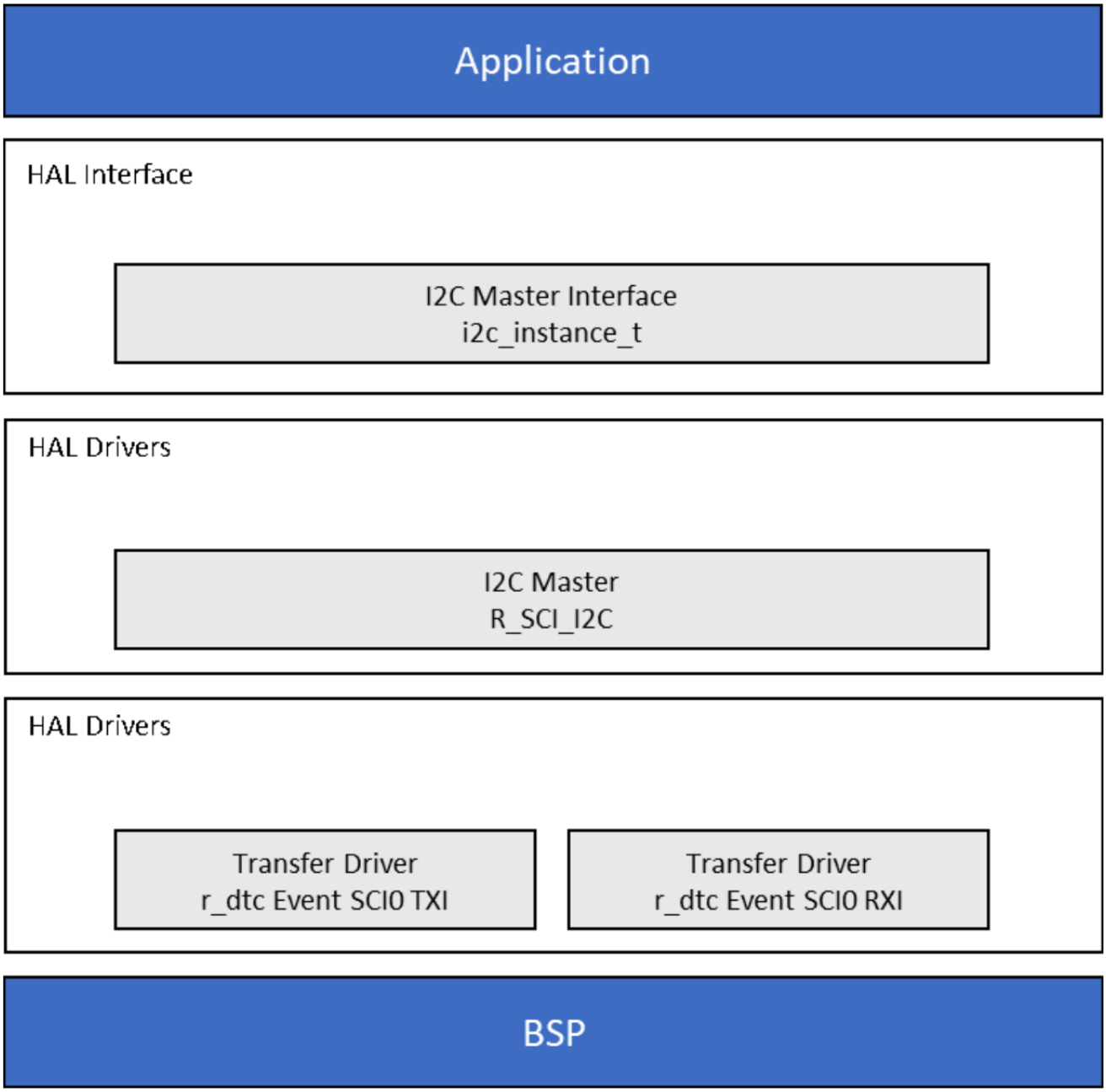


Figure 310: I2C SCI HAL Module Block Diagram

I2C SCI Hardware Support Details

The following hardware features are, or are not, supported by SSP for the I2C over SPI.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Master Mode	Slave mode	Support all Interrupt Sources	Programmable digital noise filter	Bit rate modulation	SDA delay	Timeout on bus lockout
S124	✓	N/A	ERI not supported	☒	✓	✓	N/A
S128	✓	N/A	ERI not supported	☒	✓	✓	N/A
S1JA	✓	N/A	ERI not supported	☒	✓	✓	N/A
S3A1	✓	N/A	ERI not supported	☒	✓	✓	N/A
S3A3	✓	N/A	ERI not supported	☒	✓	✓	N/A
S3A6	✓	N/A	ERI not supported	☒	✓	✓	N/A
S3A7	✓	N/A	ERI not supported	☒	✓	✓	N/A
S5D3	✓	N/A	ERI not supported	☒	✓	✓	N/A
S5D5	✓	N/A	ERI not supported	☒	✓	✓	N/A
S5D9	✓	N/A	ERI not supported	☒	✓	✓	N/A
S7G2	✓	N/A	ERI not supported	☒	✓	✓	N/A

4.2.23.2 I2C SCI HAL Module APIs Overview

The I2C SCI HAL module defines APIs for reading and writing using a master I2C device. A complete list of the available API functions, an example API function call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2C SCI HAL Module API Summary

Function Name	Example API Call and Description
open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware.
close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.
read	g_i2c.p_api->read(g_i2c.p_ctrl, &destination, bytes, restart); Performs a read operation on an I2C device.
write	g_i2c.p_api->write(g_i2c.p_ctrl, &destination, bytes, restart); Performs a write operation on an I2C device.
reset	g_i2c.p_api->reset(g_i2c.p_ctrl); Reset the peripheral.
slaveAddressSet	g_i2c.p_api->slaveAddressSet(g_i2c.p_ctrl, slave, addr_mode); Sets address of the slave device without reconfiguring the bus.
versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_RATE	The requested rate cannot be set.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device was not even opened.
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.23.3 I2C SCI HAL Module Operational Overview

The I2C SCI Master HAL module supports transactions with an I2C slave device. Callbacks are provided to interrupt the CPU when a transmission has been completed or aborted, or receive completed. The I2C SCI HAL module invokes the callback with the argument i2c_callback_args_t, indicating the number of received or transmitted bytes in buffer, pointer to user provided context, and the event i2c_event_t.

I2C SCI HAL Module Important Operational Notes and Limitations

I2C SCI HAL Module Operational Notes

Interrupts

- The I2C interrupts (SCI Error (EEI), Receive Buffer Full (RXI), Transmit Buffer Empty (TXI), and Transmit End (TEI)) for the selected channel must be enabled in the board support package (BSP), without consideration of whether the user wants to use callbacks.
- Setting the interrupts to different priority levels could result in improper operation.

IIC Rate Calculation

- The I2C SCI HAL module calculates the internal baud-rate setting based on the configured transfer rate and passes this to open. The closest possible baud rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned.

Triggering DMAC/DTC with the IIC

- DTC transfer support is added by default in the configurator, this can be removed for CPU transfer cases. The DTC is configured in the module. No user configuration is required for this.
- DMA transfer is not supported.

Triggering ELC Events with the IIC

- The I2C SCI HAL module can trigger the start of other peripherals. See the ELC User Guide for further information.

Multiple Devices on the Same Bus

When communicating with multiple slave devices on the same bus, if these slave devices have the same configuration settings the i2c_api_master_t::slaveAddressSet API function can be used to switch between slave devices without reconfiguring (no need to close and open). The control instance and bus configuration remains the same, but the slave address and addressing mode changes. A single instance of the I2C SCI HAL module is sufficient for this case.

When communicating with multiple slave devices on the same bus, if each slave device requires different configuration settings multiple instances of r_sci_i2c will be used- each configured as required for each slave device. Each instance of r_sci_i2c will be opened and closed when it is used to communicate with the target slave device.

A mix of the above two approaches can be used with a mix of slave devices. For example, if two slave devices share the same configuration settings, they can share the same instance of r_sci_i2c and the i2c_api_master_t::slaveAddressSet API function can be used to switch between them. If on the same bus, three other slave devices share the same configuration settings, but these settings are different from the first two, they will need a separate r_sci_i2c instance. Switching between each of these three slave devices can again use the i2c_api_master_t::slaveAddressSet API function. When switching between slave devices served by different instances of r_sci_i2c, the open and close technique must be used. In general, the configuration settings of the most recently slave device accessed are 'remembered' and can be re-used if appropriate. If a slave requires a different configuration than the most recently accessed slave, the r_sci_i2c module must be closed and then opened using the required configuration settings.

Applications using multiple devices connected on the same channel need to define the following macro in the pre-processor settings of the project (or the project may not build correctly):

SSP_SUPPRESS_ISR_<device_name>

Where <device_name> is the name of the additional device connected to the same channel.

I2C SCI HAL Module Limitations

- The I2C SCI HAL module in IRQ mode may not work with certain slave devices; you need to enable DTC transfer mode to work with such devices.
- To support high bit rate data transfers when operating in the IRQ mode it is highly recommended that the DTC transfer mode be enabled. This reduces the CPU overhead in responding to interrupts directly and thus removes a potential barrier in successfully implementing high bit-rate transfers.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.23.4 Including the I2C SCI HAL Module in an Application

This section describes how to include the I2C SCI HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2C Master Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2C Master Driver is g_i2c0. This name can be changed in the associated Properties window.)

I2C SCI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Master Driver on r_sci_i2c	Threads	**New Stack> Driver> Communications> I2C Master Driver on r_sci_i2c**

When the I2C Master Driver module on r_sci_i2c is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with

a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

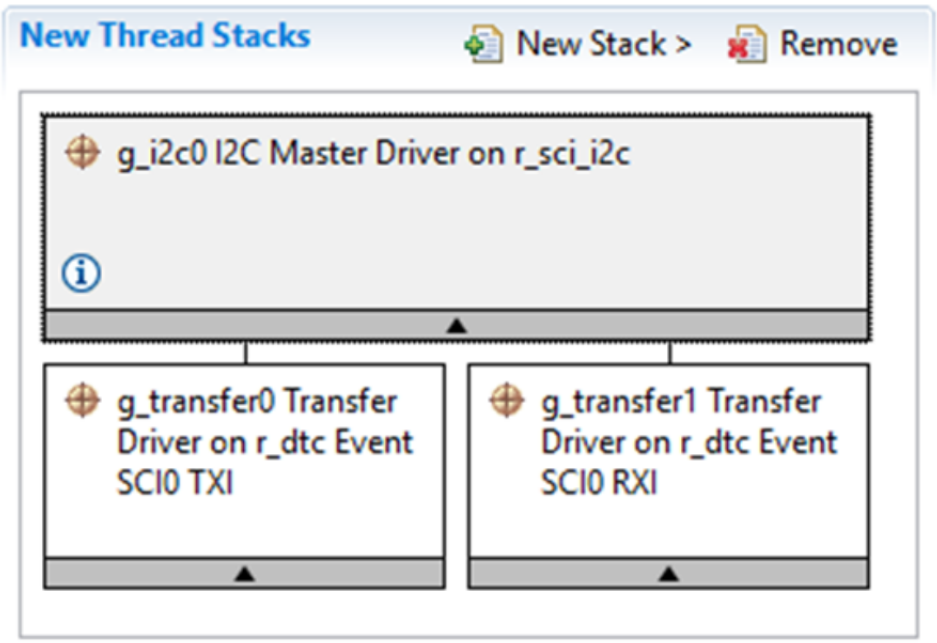


Figure 311: I2C SCI HAL Module Stack

4.2.23.5 Configuring the I2C SCI HAL Module

The I2C SCI HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C SCI HAL Module on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.

Name	g_i2c0	Module name.
Channel	0 to 9	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7 : 0 1 2 3 4 5 6 7 8 9; Series S3 : 0 1 2 3 4 - - - - 9; Series S1 : 0 1 - - - - - - - 9 .
Rate	Standard, Fast-mode Default: Standard	Select the I2C data rate.
Slave Address	0x00	Specify the slave address.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.
Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the receive interrupt priority.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit interrupt priority.

Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the transmit end interrupt priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C SCI HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer0	Module name.
Mode	Block	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Fixed	Select the transfer size.
Source Address Mode	Incremented	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Source	Select the address mode for the source.
Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.

Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Software Start	Enabled, Disabled Default: Disabled	Include code for software start in the build.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place the DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Specify the hardware channel.
Transfer Size	1 Byte	Select the transfer mode.
Destination Address Mode	Incremented	Select the transfer size.
Source Address Mode	Fixed	Select the address mode for the destination.
Repeat Area (Unused in Normal Mode	Destination	Select the address mode for the source.

Interrupt Frequency	After all transfers have completed	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source (Must enable IRQ)	Event SCI0 RXI	Select the DTC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback (Only valid with Software start)	NULL	A user callback that is called at the end of the transfer.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I2C SCI HAL Module Clock Configuration

The SCI peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate is calculated and set internally by the driver (depending on the selected transfer rate). If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error is returned when initializing the driver.

I2C SCI HAL Module Pin Configuration

The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins:

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for I2C SCI HAL Module on r_sci_i2c

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals**> **Connectivity: SCI**> **SCI0**

Note

The selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for the I2C SCI HAL Module on r_sci_i2c

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed Default: _A only	Pin group selection.
Operation Mode	Enabled, Disabled Default: Disabled	Enable or disable peripheral module.
SDA	None, P401, P407 Default: None	SDA Pin.
SCL	None, P400, P204 Default: None	SCL Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.23.6 Using the I2C SCI HAL Module in an Application

The steps in using the I2C SCI HAL module in a typical application are:

1. Initialize and open the I2C SCI HAL module using the i2c_api_master_t::open API.
2. Transfer data to the slave using the i2c_api_master_t::write API.
3. Receive data from the slave using the i2c_api_master_t::read API.
4. Operate on the received data as needed by the application.
5. Reset the module with the i2c_api_master_t::reset API (Optional).
6. Perform transactions with slave device in the application code.
7. Change the slave address using the i2c_api_master_t::slaveAddressSet API (Optional).
8. Perform transactions with slave device in the application code (Optional).
9. Close the channel using the i2c_api_master_t::close API (Optional).

These common steps are illustrated in a typical operational flow diagram in the following figure:

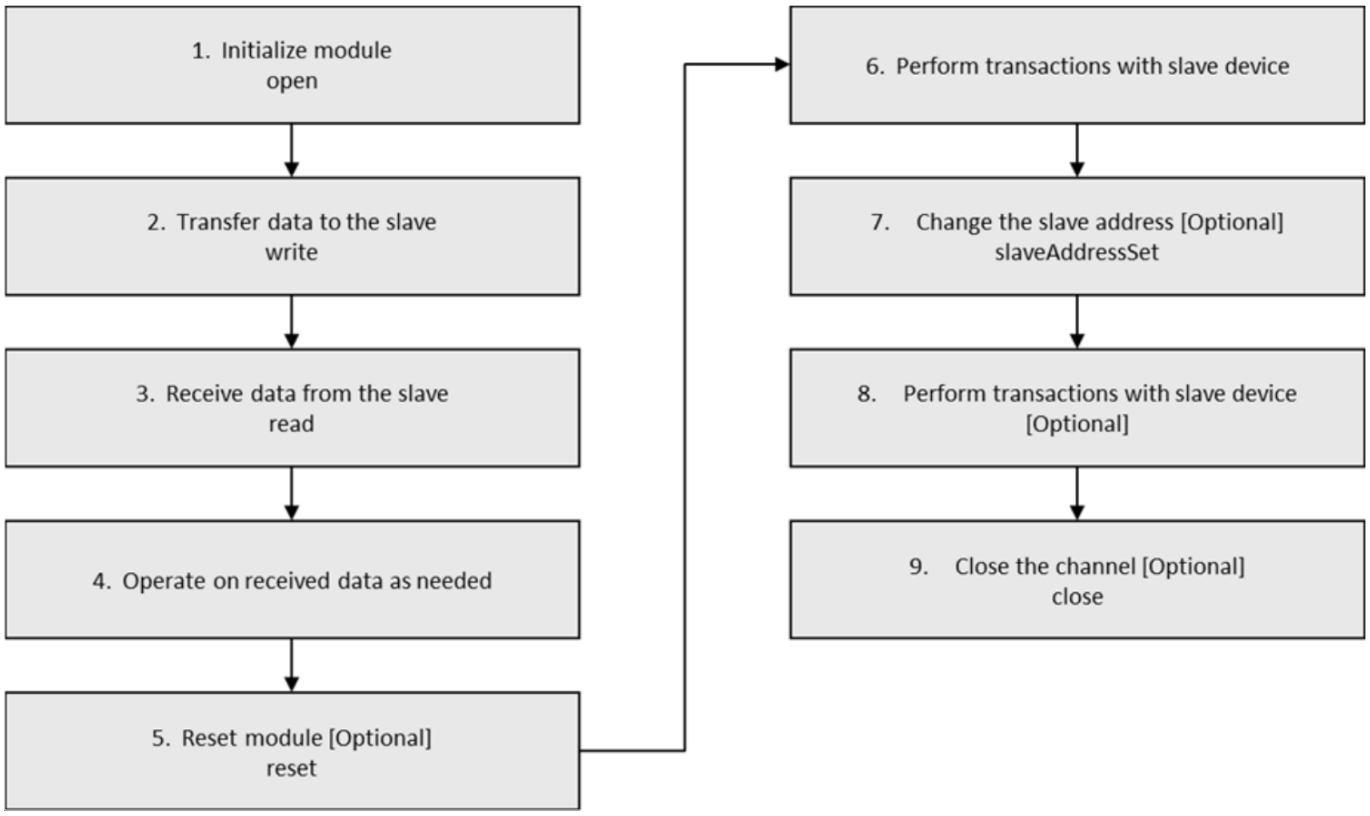


Figure 312: Flow Diagram of a Typical I2C SCI HAL Module Application

4.2.24 I2C Master Driver

4.2.24.1 I2C Master HAL Module Introduction

The I2C Master on RIIC HAL module provides a high-level API for industry standard I2C serial communications applications and uses the IIC peripheral on a Synergy MCU. Callbacks are provided for transmit complete and receive complete events notification.

I2C Master HAL Module Features

- Support for I2C RIIC operations
  - Standard (up to 100 kHz)
  - I2C fast-mode (up to 400 kHz)
  - I2C fast-mode plus (up to 1 MHz on channel 0 (SCL0-A, SDA0-A) of S7G2 and S5D9 MCU families)
- Initialization of the RIIC module
- Read from a slave device
- Write to a slave device
- Reset the MCUs I2C peripheral
- Set the address of the slave device
- Callback support
  - Transfer aborted (along with exact IIC hardware-generated error event)

- Transmit complete (number of bytes transmitted provided)
- Receive complete (number of bytes received provided)

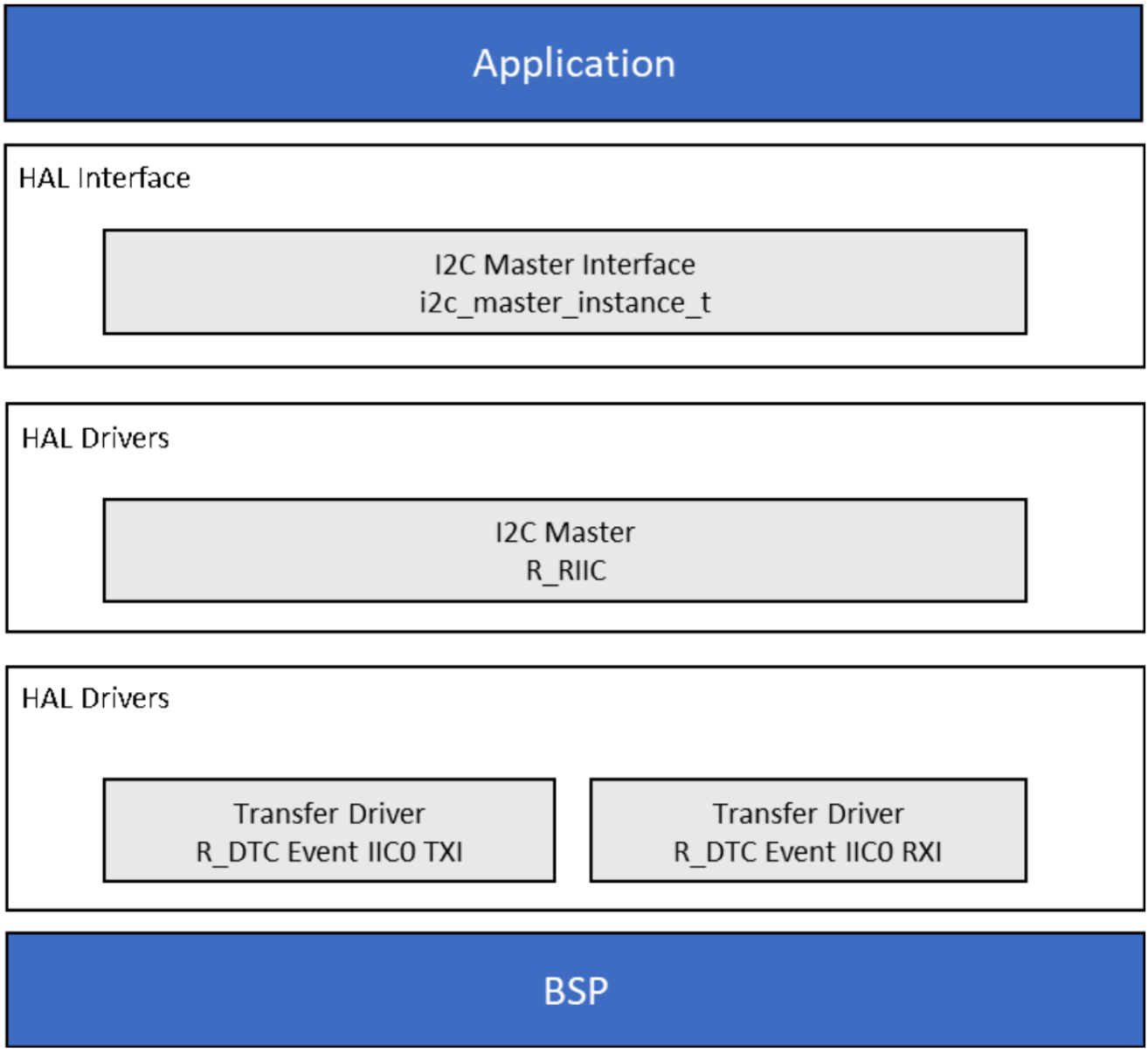


Figure 313: I2C Master HAL Module Block Diagram

RIIC Master Hardware support details

The following hardware features are, or are not, supported by SSP for the RIIC Master Driver:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	I2C Format	SMBus Format	Master Mode	Slave Mode	Fast Mode Plus	Selectable duty cycle
S124	✓	☒	✓	✓	N/A	☒
S128	✓	☒	✓	✓	N/A	☒
S1JA	✓	☒	✓	✓	N/A	☒
S3A1	✓	☒	✓	✓	N/A	☒
S3A3	✓	☒	✓	✓	N/A	☒
S3A6	✓	☒	✓	✓	N/A	☒
S3A7	✓	☒	✓	✓	N/A	☒
S5D3	✓	☒	✓	✓	✓	☒
S5D5	✓	☒	✓	✓	✓	☒
S5D9	✓	☒	✓	✓	✓	☒
S7G2	✓	☒	✓	✓	✓	☒

MCU Group	Configurable to up to three different slave addresses	7- and 10-bit address formats	General call address, Device ID address and SMBus host address detectable	Automatic loading of the acknowledge bit	SDA output delay function	Selectable Wait functions (8/9 or 9/1)
S124	☒	7 and 10-bit	☒	✓	✓	☒
S128	☒	7 and 10-bit	☒	✓	✓	☒
S1JA	☒	7 and 10-bit	☒	✓	✓	☒
S3A1	☒	7 and 10-bit	☒	✓	✓	☒
S3A3	☒	7 and 10-bit	☒	✓	✓	☒
S3A6	☒	7 and 10-bit	☒	✓	✓	☒
S3A7	☒	7 and 10-bit	☒	✓	✓	☒
S5D3	☒	7 and 10-bit	☒	✓	✓	☒
S5D5	☒	7 and 10-bit	☒	✓	✓	☒
S5D9	☒	7 and 10-bit	☒	✓	✓	☒
S7G2	☒	7 and 10-bit	☒	✓	✓	☒

MCU Group	Full Arbitration Support	Internal Detect Time-Out	Programmable Digital Noise Filters	Support all Interrupt Sources
S124	Master, NACK arbitrations	✓	☒	✓
S128	Master, NACK arbitrations	✓	☒	✓

S1JA	Master, NACK arbitrations	✓	☒	✓
S3A1	Master, NACK arbitrations	✓	☒	✓
S3A3	Master, NACK arbitrations	✓	☒	✓
S3A6	Master, NACK arbitrations	✓	☒	✓
S3A7	Master, NACK arbitrations	✓	☒	✓
S5D3	Master, NACK arbitrations	✓	☒	✓
S5D5	Master, NACK arbitrations	✓	☒	✓
S5D9	Master, NACK arbitrations	✓	☒	✓
S7G2	Master, NACK arbitrations	✓	☒	✓

4.2.24.2 I2C Master HAL Module APIs Overview

The I2C Master on RIIC (I2C RIIC) HAL module defines API functions including reading and writing using a master I2C device. A complete list of the available API functions, an example API function call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2C Master HAL Module API Summary

Function Name	Example API Call and Description
open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware.
close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.
read	g_i2c.p_api->read(g_i2c.p_ctrl, &destination, bytes, restart); Performs a read operation on an I2C device.
write	g_i2c.p_api->write(g_i2c.p_ctrl, &destination, bytes, restart); Performs a write operation on an I2C device.
reset	g_i2c.p_api->reset(g_i2c.p_ctrl); Reset the peripheral.
versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

slaveAddressSet	g_i2c.p_api->slaveAddressSet(g_i2c.p_ctrl, slave_addr, addr_mode); Reconfigures the slave address.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_POINTER	Pointer is NULL.
SSP_ERR_IN_USE	I2C bus busy state detected during I2C transaction operation or attempted to open an already open device instance only during open API call.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INVALID_RATE	The requested rate cannot be set.
SSP_ERR_HW_LOCKED	Driver busy doing riic operation.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.24.3 I2C Master HAL Module Operational Overview

The I2C master on RIIC HAL module supports transactions with an I2C Slave device. Callbacks are provided to interrupt the CPU when a transmission or receive has been completed. The RIIC HAL module invokes the callback with the argument i2c_callback_args_t, indicating the number of received or transmitted bytes in buffer, pointer to user provided context, and the event i2c_event_t.

I2C Master HAL Module Important Operational Notes and Limitations

I2C Master HAL Module Operational Notes

Interrupts

- The RIIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device irrespective of whether the user wants to use callbacks.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

IIC Rate Calculation

- The I2C Master module calculates the internal baud-rate setting based on the configured

transfer rate and passed to open. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned.

Triggering DMAC/DTC with the IIC

- DTC transfer support added by default in the configurator. This can be removed for CPU transfer cases. The DTC is configured in the module. No user configuration is required for this.
- DMA transfer is not supported.

Triggering ELC Events with the IIC

- The I2C Master module can trigger the start of other peripherals. See events and peripheral definitions in the ELC User Guide for further information.

Multiple Devices on the Bus

- If multiple devices are connected on the same bus, only one device can be opened at a time.
- If multiple slave devices are on the same bus, and they have different configurations, the application program should use multiple I2C master modules- one for each configuration.
- If the application wants to switch the device without opening and closing the bus, use the i2c_api_master_t::slaveAddressSet API where g_i2c.p_ctrl is the same control instance that was used in the last opened device. The module will use the same bus configuration to communicate with the new device when the application program subsequently calls the i2c_api_master_t::read or i2c_api_master_t::write API functions.

Usage of Restart Condition

- Passing the value 'true' to the restart parameter of the write/read API will generate restart condition after specified number (length) of bytes.The master will continue to hold the bus busy (low) without timeout so that current master can trigger the next write/read API.

Multi-Master Support

If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

SDA Delay Support

The SDA Delay function delays SDA output from the detection of a falling edge of SCL signal to ensure that

the SDA signal is output within the interval during which the SCL is low.

I2C Master HAL Module Limitations

- Any of the supported IIC channel can be configured for either Master or Slave mode operation but not for both.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.24.4 Including the I2C Master HAL Module in an Application

This section describes how to include the I2C Master HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2C Master Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2C Master Driver is g_i2c0. This name can be changed in the associated Properties window.)

I2C Master HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Master Driver on r_riic	Threads	**New Stack> Driver> Communications> I2C Master Driver on r_riic**

When the I2C Master HAL module on r_riic is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

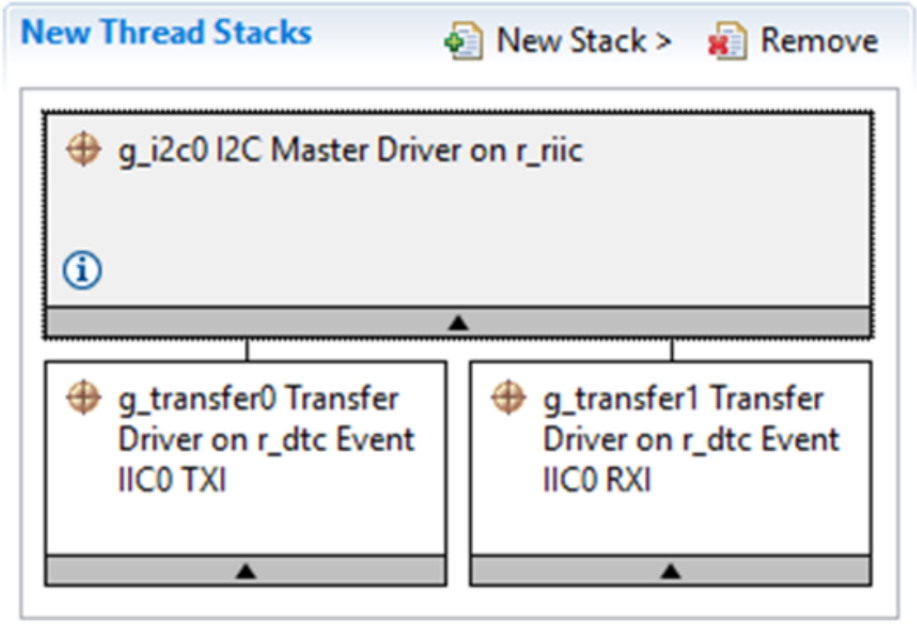


Figure 314: I2C Master HAL Module Stack

4.2.24.5 Configuring the I2C Master HAL Module

The I2C Master HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C Master HAL Module on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0	Specify the IIC channel to be used with this configuration.
Rate	Standard, Fast-mode, Fast-mode Plus Default: Standard	Standard, Fast, and Fast-plus. (See IIC Rate Calculation.)
Slave Address	0x00	Set the address of the slave device the I2C master will be communicating with.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay(nanoseconds)	Default: 300	SDA output delay in nanoseconds.

Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t*. *Exact hardware generated error event is also provided when i2c_event_t is I2C_EVENT_ABORTED. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Error interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 TXI

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event IIC0 TXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.

Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I2C Master HAL Module Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set internally by the driver depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned when initializing the driver.

I2C Master HAL Module Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the

method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins:

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for I2C Master HAL Module on r_riic

Resource	ISDE Tab	Pin selection Sequence
IIC	Pins	Select **Peripherals**> **Connectivity: IIC**> **IIC0**

Note

The selection sequence assumes IIC0 is the desired hardware target for the driver.

Pin Configuration Settings for the I2C Master HAL Module on r_sci_uart

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed Default: _A only	Pin group selection.
Operation Mode	Enabled, Disabled Default: Disabled	Enable or disable peripheral module.
SDA	None, P401, P407 Default: None	SDA Pin.
SCL	None, P400, P204 Default: None	SCL Pin.
SCK	None, P412, P102 Default: P412	SCK Pin.
CTS_RTS_SS	None, P413, P103 Default: None	CTS Pin.
SDA	Disabled	SDA Pin (when Simple I2C is used).
SCL	Disabled	SCL Pin (when Simple I2C is used).

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.24.6 Using the I2C Master HAL Module in an Application

The steps in using the I2C Master HAL module in a typical application are:

1. Initialize and open the I2C RIIC HAL Module using the i2c_api_master_t::open API.
2. Transfer data to the slave using the i2c_api_master_t::write API.
3. Receive data from the slave using the i2c_api_master_t::read API.
4. Reset the module using the i2c_api_master_t::reset API. (Optional)
5. Change the slave address using i2c_api_master_t::slaveAddressSet API. (Optional)

6. Transfer data to the slave using the i2c_api_master_t::write API. (Optional)
7. Receive data from the slave using the i2c_api_master_t::read API. (Optional)
8. Close the module using the i2c_api_master_t::close API. (Optional)

These common steps are illustrated in a typical operational flow diagram in the following figure:

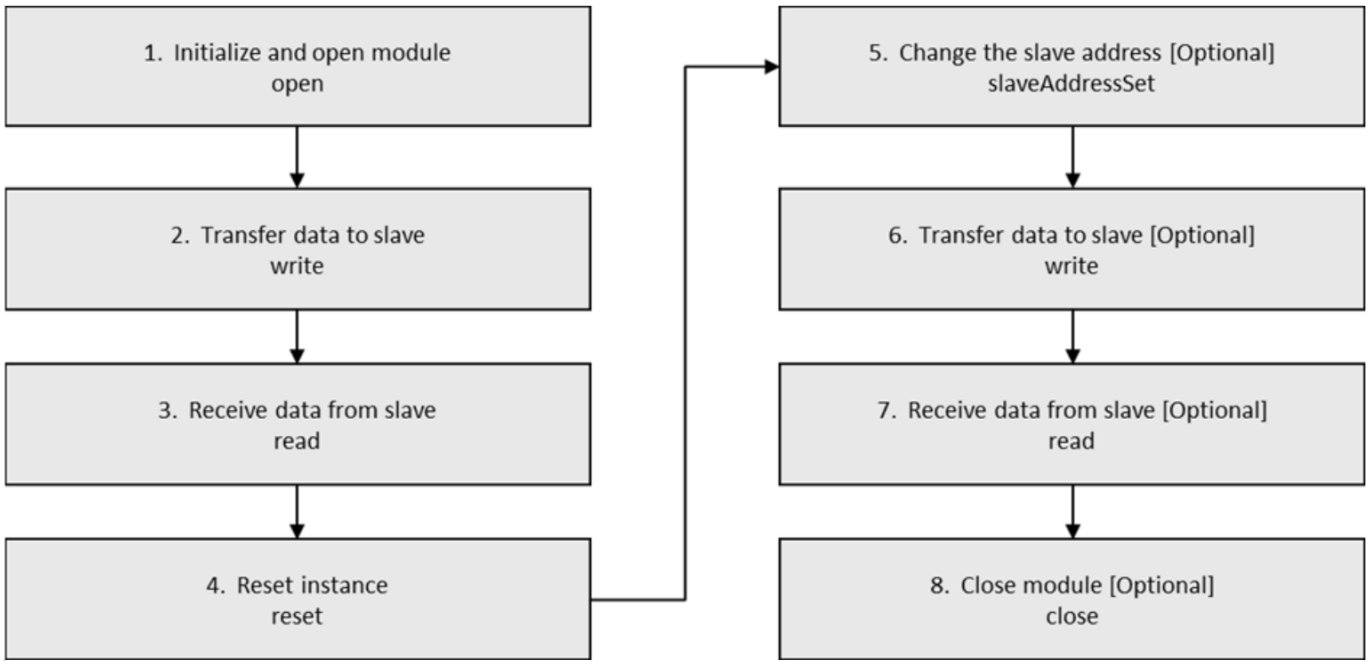


Figure 315: Flow Diagram of a Typical I2C Master HAL Module Application

4.2.25 I2C Slave Driver

4.2.25.1 I2C Slave HAL Module Introduction

The I2C Slave HAL Module provides a high-level API for I2C slave applications and uses the RIIC peripheral on the Synergy MCU. Callbacks are provided to signal read/write request received from master and transfer completion events.

I2C Slave HAL Module Features

- Support for I2C Slave operations
- Support transactions with a I2C master device
  - Read
  - Write
- Callback support
  - Transmit Request (notifies when a write operation is expected from slave)
  - Receive Request (notifies when a read operation is expected from slave)
  - Transmit more request (notifies when master requests more data than configured in slave write operation. Also provides number of bytes transmitted)
  - Receive more request (notifies when master tries to write more data than configured in slave read operation. Also provides number of bytes received)

- Transmit complete (provides number of bytes transmitted)
- Receive complete (provides number of bytes received)

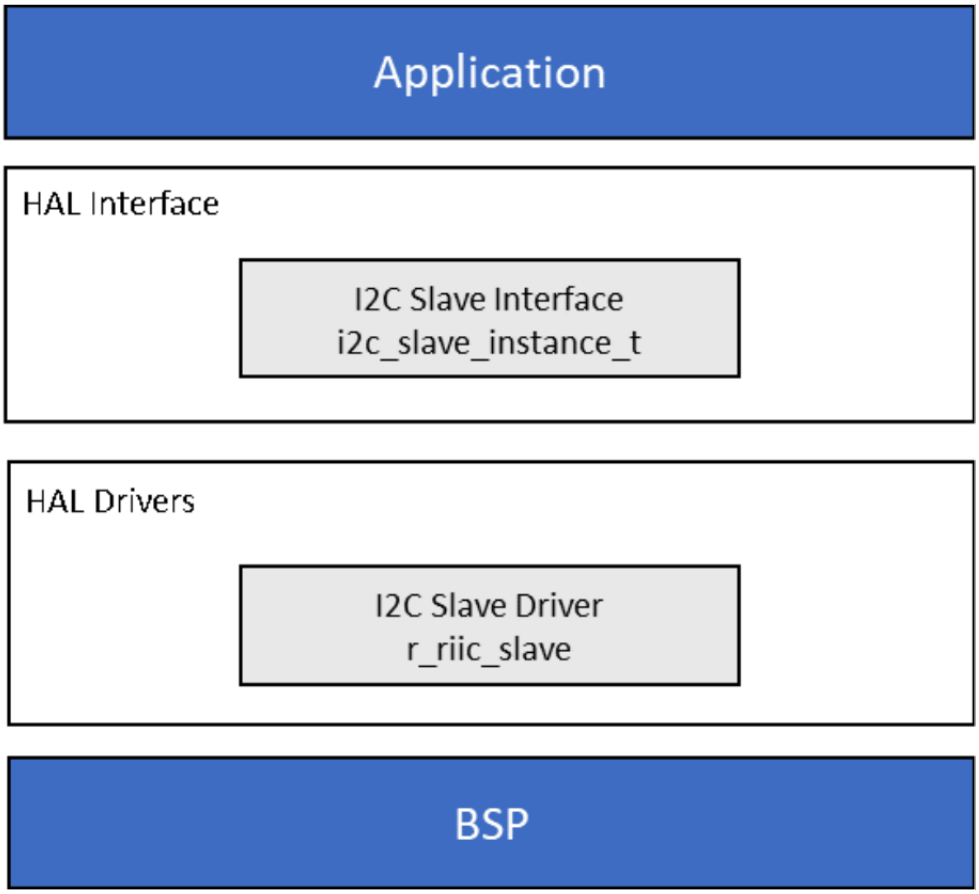


Figure 316: I2C Slave HAL Module Block Diagram

RIIC Slave Hardware support details

RIIC Slave driver supported features:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	I2C Format	SMBus Format	Master Mode	Slave Mode	Fast Mode Plus	Selectable duty cycle
S124	✓	☒	☒	✓	N/A	☒
S128	✓	☒	☒	✓	N/A	☒

S1JA	✓	☒	☒	✓	N/A	☒
S3A1	✓	☒	☒	✓	N/A	☒
S3A3	✓	☒	☒	✓	N/A	☒
S3A6	✓	☒	☒	✓	N/A	☒
S3A7	✓	☒	☒	✓	N/A	☒
S5D3	✓	☒	☒	✓	✓	☒
S5D5	✓	☒	☒	✓	✓	☒
S5D9	✓	☒	☒	✓	✓	☒
S7G2	✓	☒	☒	✓	✓	☒

MCU Group	Configurable to up to three different slave addresses	7- and 10-bit address formats	General call address, Device ID address and SMBus host address detectable	Automatic loading of the acknowledge bit	SDA output delay function	Selectable Wait functions (8/9 or 9/1)
S124	One slave address	7 and 10-bit	☒	✓	☒	☒
S128	One slave address	7 and 10-bit	☒	✓	☒	☒
S1JA	One slave address	7 and 10-bit	☒	✓	☒	☒
S3A1	One slave address	7 and 10-bit	☒	✓	☒	☒
S3A3	One slave address	7 and 10-bit	☒	✓	☒	☒
S3A6	One slave address	7 and 10-bit	☒	✓	☒	☒
S3A7	One slave address	7 and 10-bit	☒	✓	☒	☒
S5D3	One slave address	7 and 10-bit	☒	✓	☒	☒
S5D5	One slave address	7 and 10-bit	☒	✓	☒	☒
S5D9	One slave address	7 and 10-bit	☒	✓	☒	☒
S7G2	One slave address	7 and 10-bit	☒	✓	☒	☒

MCU Group	Full Arbitration Support	Internal Detect Time-Out	Programmable Digital Noise Filters	Support all Interrupt Sources	Event link function through ELC HAL driver
S124	Slave, NACK arbitration	✓	☒	✓	☒
S128	Slave, NACK arbitration	✓	☒	✓	☒
S1JA	Slave, NACK arbitration	✓	☒	✓	☒
S3A1	Slave, NACK arbitration	✓	☒	✓	☒
S3A3	Slave and NACK arbitration	✓	☒	✓	☒
S3A6	Slave and NACK arbitration	✓	☒	✓	☒
S3A7	Slave, NACK arbitration	✓	☒	✓	☒
S5D3	Slave, NACK arbitration	✓	☒	✓	☒
S5D5	Slave, NACK arbitration	✓	☒	✓	☒
S5D9	Slave, NACK arbitration	✓	☒	✓	☒
S7G2	Slave and NACK arbitration	✓	☒	✓	☒

4.2.25.2 I2C Slave HAL Module APIs Overview

The I2C RIIC Slave HAL Module defines APIs for reading and writing to a master I2C device. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2C Slave HAL Module API Summary

Function Name	Example API Call and Description
open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware.
close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.

masterWriteSlaveRead	g_i2c.p_api->masterWriteSlaveRead(g_i2c.p_ctrl, &destination, bytes); Performs a read operation on an I2C device.
masterReadSlaveWrite	g_i2c.p_api->masterReadSlaveWrite(g_i2c.p_ctrl, &source, bytes); Performs a write operation on an I2C device.
versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_POINTER	Pointer is NULL.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.25.3 I2C Slave HAL Module Operational Overview

The I2C RIIC Slave HAL Module supports transfers to an I2C Master device. Callbacks are provided to enable the application identify the occurrence of any of the following events.

- Read/write request with the slave address match is detected from master.
- Master requests more data than slave configured in read/write API.
- An ongoing transfer has been aborted.
- An ongoing transfer has completed.

I2C Slave HAL Module Important Operational Notes and Limitations

I2C Slave HAL Module Operational Notes

- The RIIC Error (EEI), Receive Buffer Full (RXI), Transmit Buffer Empty (TXI) and Transmit End (TEI) interrupts for the selected channel used must be enabled in the BSP irrespective of whether the user wants to use callbacks.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.
- If using RIIC and RIIC_Slave modules on the same board, it is suggested to set equal interrupt priority for TXI, TEI and EI and set RXI interrupt priority higher than these.

I2C Slave HAL Module Limitations

This is the initial version of I2C RIIC Slave Driver with only basic functionality implemented. The following limitations are known:

1. For the driver provide to the application any information regarding the type of request received, the application has to implement/register for the appropriate callback. This limits in usage of the events provided by the driver in application. These events help in real-time master request processing.
2. When the driver is used in blocking mode, slave API has to be invoked with same number of bytes as configured in the master.
3. Any of the supported IIC channels can be configured for either Master or Slave mode operation but not for both.

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.25.4 Including the I2C Slave HAL Module in an Application

This section describes how to include the I2C Slave HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2C Slave Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2C Slave Driver is g_i2c0. This name can be changed in the associated Properties window.)

I2C Slave HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Slave Driver on r_riic_slave	Threads	**New Stack> Driver> Communications> I2C Slave Driver on r_riic_slave**

When the I2C Slave Driver on r_riic_slave is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Grayband are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

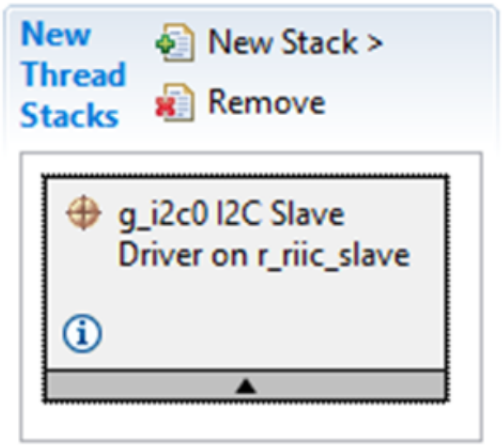


Figure 317: I2C Slave HAL Module Stack

4.2.25.5 Configuring the I2C Slave HAL Module

The I2C Slave HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C Slave HAL Module on r_riic_slave

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0	Specify the IIC channel to be used with this configuration.
Rate	Standard, Fast-mode, Fast-mode plus Default: Standard	Transfer rate to which the IIC peripheral should be configured to operate.
Slave Address	0x00	Set the address of the device as the I2C slave address. Both 7-bit and 10-bit addresses are supported.

Address Mode	7-Bit, 10 Bit Default: 7-Bit	Address mode selection.
Callback	NULL	A user callback function can be registered in i2c_api_master_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit End interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I2C Slave HAL Module Clock Configuration

The RIIC peripheral module uses PCLKB as its clock source.

I2C Slave HAL Module Pin Configuration

The RIIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins:

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the I2C Slave HAL Module on r_riic_slave

Resource	ISDE Tab	Pin selection Sequence
I2C	Pins	Select **Peripherals**> **Connectivity: IIC**> **IIC0.**

Note

The selection sequence assumes that IIC0 is the desired hardware target for the driver.

Pin Configuration Settings for the I2C Slave HAL Module on r_riic_slave

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed Default: _A only	Select Simple I2C as the Operation Mode for I2C on SCI.
Operation Mode	Enabled, Disabled Default: Disabled	Enable or disable peripheral module.
SDA	None, P401, P407 Default: None	SDA Pin.
SCL	None, P400, P204 Default: None	SCL Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and SK-S7G2 Kit. Other Synergy kits and MCUs may have different available pin configuration settings.

4.2.25.6 Using the I2C Slave HAL Module in an Application

The sequence for using the I2C Slave HAL module (blocking mode) in an application is:

1. Initialize and open the I2C Slave HAL Module using the i2c_api_slave_t::open API.

2. Transfer data to the master using the i2c_api_slave_t::masterReadSlaveWrite API.

3. Receive data from the master using the i2c_api_slave_t::masterWriteSlaveRead API.

4. Close the channel using the i2c_api_slave_t::close API.

These steps are illustrated in a typical operational flow diagram in the following figure:

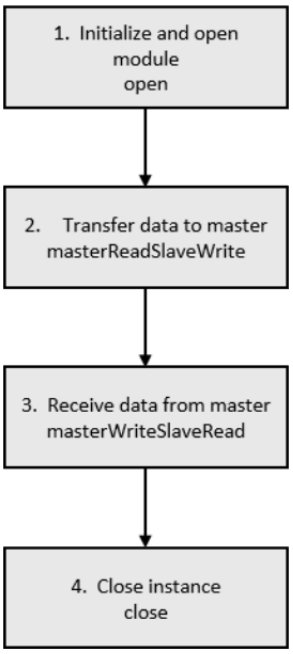


Figure 318: Flow Diagram of a Typical I2C Slave HAL Module Application without callback

The sequence for using the I2C Slave HAL module (non-blocking mode) in an application is:

1. Initialize and open the I2C Slave HAL Module using the i2c_api_slave_t::open API.
2. If callback event i2c_event_t generated indicates transmit request. Transfer data to the master using the i2c_api_slave_t::masterReadSlaveWrite API in callback.
3. If callback event i2c_event_t generated indicates transmit-more request (master tries to read more data than configured in slave write). Transfer data to the master using the i2c_api_slave_t::masterReadSlaveWrite API in callback with new transmit data buffer.
4. If callback event i2c_event_t generated indicates receive request. Receive data from the master using the i2c_api_slave_t::masterWriteSlaveRead API in callback.
5. If callback event i2c_event_t generated indicates a receive-more request (master tries to write more data than configured in slave read). Receive data from the master using the i2c_api_slave_t::masterWriteSlaveRead API in callback with new receive data buffer.
6. Close the channel using the i2c_api_slave_t::close API.

These steps above with registered callback are illustrated in a typical operational flow diagram in the following figure:

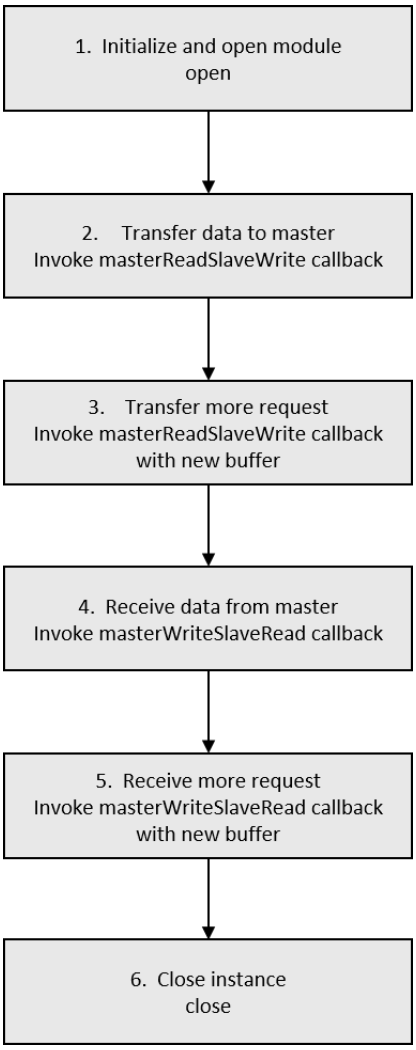


Figure 319: Flow Diagram of a Typical I2C Slave HAL Module Application with callback

4.2.26 I2S Driver

4.2.26.1 I2S HAL Module Introduction

The I2S HAL module provides a high-level API for the standard I2S audio serial communication protocol used to send or receive uncompressed audio data in master/slave mode.

I2S HAL Module Features

The I2S HAL module used with the SSI peripheral in I2S master/slave mode supports the following features (in addition to the standard I2S protocol):

- Full-duplex I2S communication (SSI channel 0 only)
- Interrupt driven data transmission and reception
- Integration with the DTC transfer module

- A user-defined callback created to respond to the need for additional data

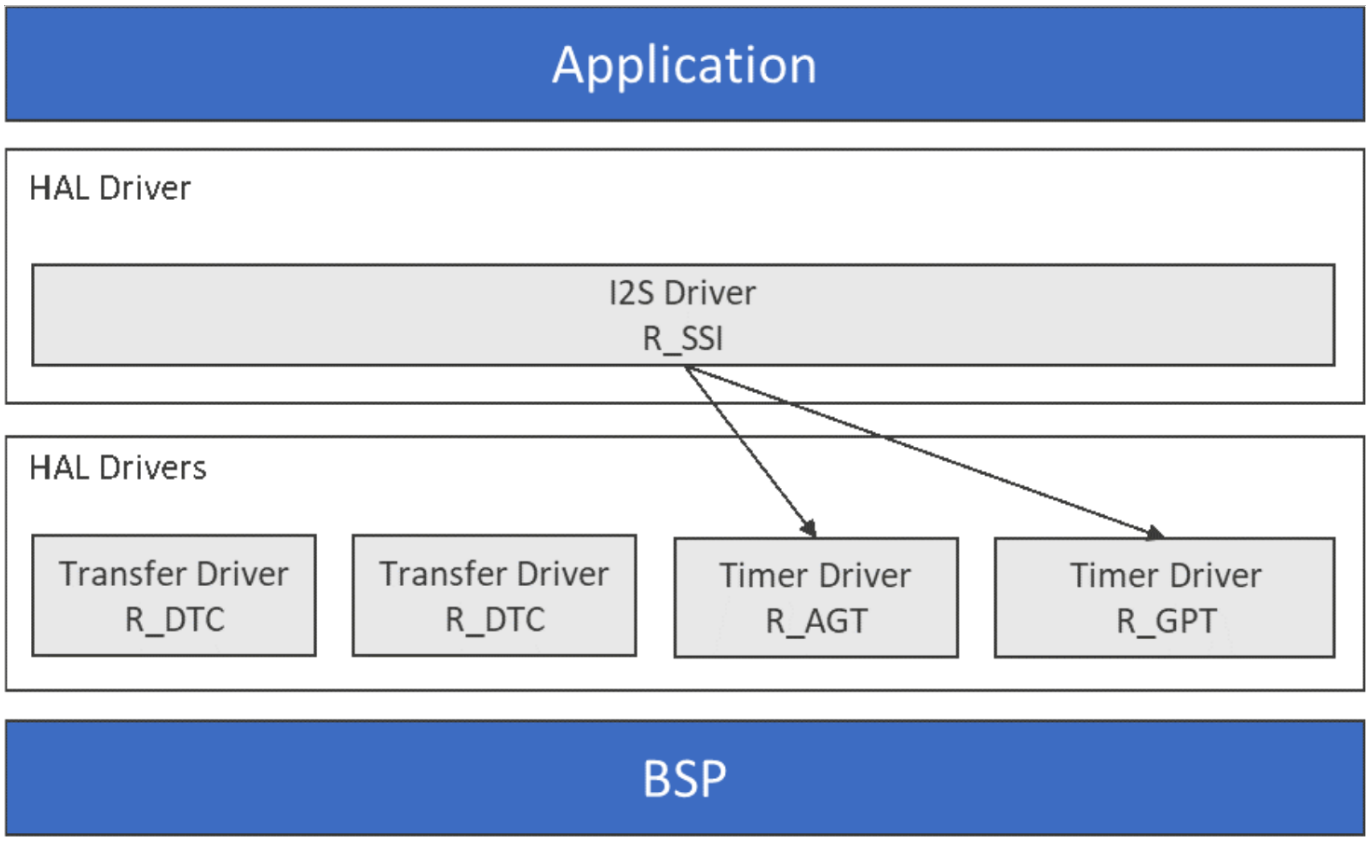


Figure 320: I2S HAL Module Block Diagram

I2S Hardware Support Details

The following hardware features are, or are not, supported by SSP for I2S:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Supports 2 Channels	SSI Format Support	MSB first Format Support	Serial bit clock configurable {16, 32, 48, and 64 sampling rates}	Master clock input from the master clock pin for audio (AUDIO_CLK)	Master clock input from the master clock pin for GPT output (GTIOC1A)

S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	☒	☒	☒	✓	✓	☒
S3A3	☒	☒	☒	✓	✓	☒
S3A6	☒	☒	☒	✓	✓	☒
S3A7	✓	☒	☒	✓	✓	☒
S5D3	✓	☒	☒	✓	✓	☒
S5D5	☒	☒	☒	✓	✓	☒
S5D9	✓	☒	☒	✓	✓	☒
S7G2	✓	☒	☒	✓	✓	☒

MCU Group	Ability to select Stop Word SSIWS	Accepts Interrupts from Communication Errors	Accepts Interrupts from Receive Data Full	Accepts Interrupts from Transmit Data Empty	Internal Connection to GPT output GTIOC1A
S124	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A
S3A1	☒	✓	✓	✓	✓
S3A3	☒	✓	✓	✓	✓
S3A6	☒	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓
S5D3	☒	✓	✓	✓	✓
S5D5	☒	✓	✓	✓	✓
S5D9	☒	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓

4.2.26.2 I2S HAL Module APIs Overview

The I2S HAL module defines APIs for operations such as opening, muting, writing and reading. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2S HAL Module API Summary

Function Name	Example API Call and Description
open	g_i2s0.p_api->open(g_i2s0.p_ctrl, g_i2s0.p_cfg); Initial configuration.

stop	g_i2s0.p_api->stop(g_i2s0.p_ctrl, direction_to_stop); Stop communication. Transmission is stopped when callback is called with I2S_EVENT_IDLE. Reception is stopped when callback is called with I2S_EVENT_RX_EMPTY.
mute	g_i2s0.p_api->mute(g_i2s0.p_ctrl, mute_enable); Enable or disable mute.
write	g_i2s0.p_api->write(g_i2s0.p_ctrl, &data, bytes); Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.
read	g_i2s0.p_api->read(g_i2s0.p_ctrl, &data, bytes); Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.
writeRead	g_i2s0.p_api->writeRead(g_i2s0.p_ctrl, &source, &destination, bytes); Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.
infoGet	g_i2s0.p_api->infoGet(g_i2s0.p_ctrl, &info); Get instance specific information and store it in provided pointer info.
close	g_i2s0.p_api->close(g_i2s0.p_ctrl); Allows driver to be reconfigured and may reduce power consumption.
versionGet	g_i2s0.p_api->versionGet(&version); Get version and store it in provided pointer version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_ASSERTION	A critical assertion has failed.

SSP_ERR_IN_USE	Channel is running/busy.
SSP_NOT_OPEN	Requested channel is not configured or API not open.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.26.3 I2S HAL Module Operational Overview

The I2S HAL module supports audio communications using the I2S protocol. The driver supports the SSI Peripheral on a Synergy MCU in I2S master/slave mode. It can send and receive uncompressed audio data. It provides full-duplex I2S communication (SSI channel 0 only), interrupt-driven data transmission, reception, and integration with the DTC transfer module.

I2S HAL Module Important Operational Notes and Limitations

I2S HAL Module Operational Notes

To enable audio data reception on channel 0, enable the SSI0 RXI interrupt. To enable audio data transmission on channel 0, enable the SSI0 TXI interrupt. To enable both transmission and reception on channel 0, enable both the SSI0 TXI and SSI0 RXI interrupts. To enable transmission or reception on channel 1, enable the SSIn TXI RXI interrupt. In all cases, enable the SSIn INT interrupt.

When the interrupts are enabled in the BSP, the corresponding ISRs will be defined in the I2S driver. The ISR will call the user-callback function registered in open.

I2S HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.26.4 Including the I2S HAL Module in an Application

This section describes how to include the I2S HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2S Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2S Driver is g_i2s0. This name can be changed in the associated Properties window.)

I2S HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2s0 I2S Driver on r_ssi	**Threads**> **HAL/Common Stacks**	**New Stack**> **Driver**> **Connectivity**> **I2S Driver on r_ssi**

When the I2S Driver on r_ssi is added to the thread stack as shown in the following figure, the

configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

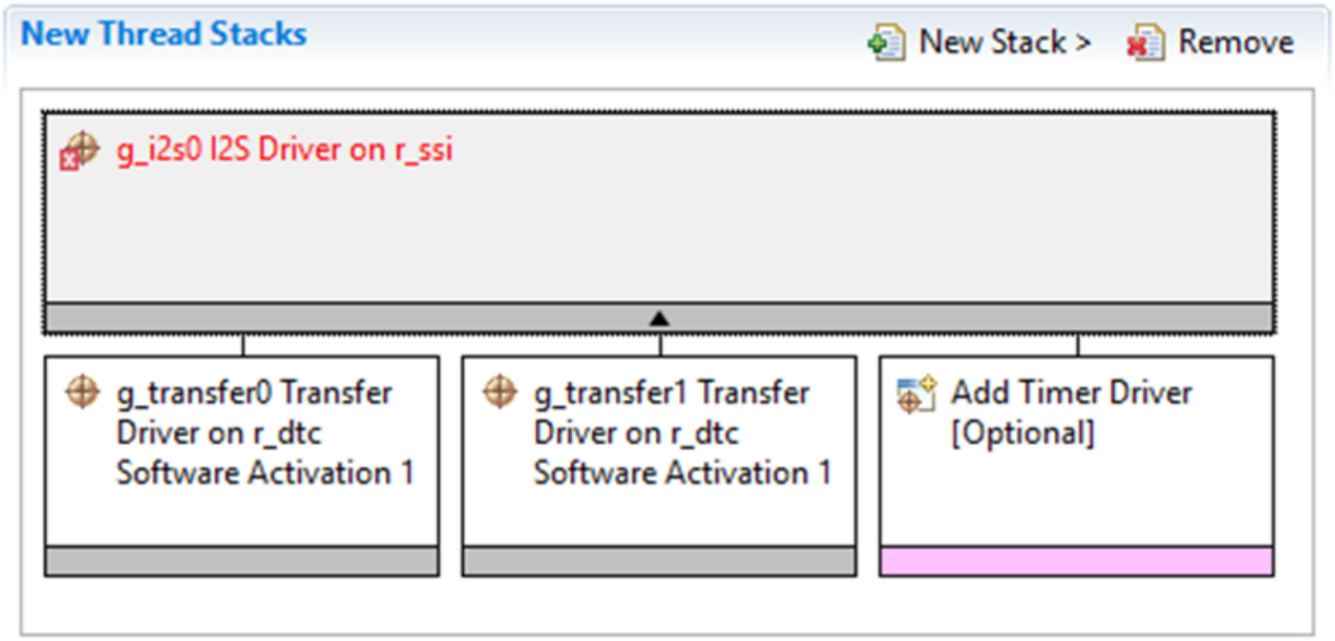


Figure 321: I2S HAL Module Stack

4.2.26.5 Configuring the I2S HAL Module

The I2S HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits)
Sampling Frequency (Hertz)	44100	Sampling frequency of audio data.
Operating Mode	Master Mode, Slave Mode Default: Master Mode	Operating mode of communication (Master/Slave).
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system

Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Transmit interrupt priority selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Receive interrupt priority selection
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Priority 12	Idle/error interrupt priority selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, it might be useful to select the DAC or I2S Channel based on the target hardware implementation. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the I2S HAL Module Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configurable Settings for the Transfer Driver on r_dtc Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.

Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC software event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.

Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC software event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configurable Settings for the Timer Driver on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.

Period Unit	Hertz	Period unit selection.
Auto Start	FALSE	Auto start selection.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	Count source selection.
AGTO Output Enabled	True, False Default: False	AGTO output selection.
AGTIO Output Enabled	True, False Default: False	AGTIO output selection.
Output Inverted	True, False Default: False	Output inverted selection.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection.
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection.
Callback	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configurable Settings for the Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Mode selection.
Period Value	2822400 *2	Period value selection.

Period Unit	Hertz	Period unit selection.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection.
Auto Start	FALSE	Auto start selection.
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection.
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection.
Callback	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I2S HAL Module Clock Configuration

The SSI module uses the peripheral clock (PCLKB) available from the Clock configuration window. It also uses an external clock input to the AUDIO_CLK pin.

I2S HAL Module Pin Configuration

The SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device.

Configure the SSI RX and/or TX pins for the selected SSI channel (**Pins** tab > **Peripherals** > **SSI** > **SSIn** > **SSITXD0/SSIRXD0**). For channel 0, enable one or both of these pins. For channel 1, enable the SSIDATA1 pin.

Configure the word select and clock pins (**Pins** tab > **Peripherals** > **SSI** > **SSIn** > **SSITWSn** and **SSISCKn**). Both of these pins are required in most cases. Consult the datasheet of the I2S device

used for the required pins.

Configure the audio clock pin (**Pins** tab > **Peripherals** > **SSI** > **SSI0_SSI1_AUDIO_CLK**) for SSI. Connect an external audio clock to this clock input pin. If a GPT timer is used to generate the audio clock, configure the GPT timer output pin (**Pins** tab > **Peripherals** > **GPT1** > **GPTn** > **GTIOCx**) and connect the GPT output pin used to the audio clock input pin.

The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins:

Pin Selection for the I2S HAL Module on r_ssi

Resource	ISDE Tab	Pin selection Sequence
ACMPLP	Pins	Select **Peripherals**> **Analog:ACMP.**

Note

The selection sequence assumes SSI0 is the desired hardware target for the driver.

Pin Configurable Settings for the I2S Driver on SSI

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, Mixed	Pin group for I2S port.
Operation Mode	Custom, Disabled	Operation selection.
SSISCK	None, P400 Default: None	AUDIO_CLK pin(P400), used in this application

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Pin Configuration Settings for I2S Driver on SSI0

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port.
Operation Mode	Enabled, Custom, Disabled	Operation selection.
SSISCK	None, P112 Default: None	SSISCK pin (P112), used in this application.
SSIWS	None, P113 Default: None	SSIWS pin (P113), used in this application.
SSITXD	None, P115 Default: None	SSITXD pin (P115), used in this application.

SSIRXD	None, P114 Default: None	SSIRXD pin (P114), used in this application.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Pin Configuration Settings for I2S Driver on SSI1

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port.
Operation Mode	Enabled, Custom, Disabled	Operation selection.
SSISCK	None, P204 Default: None	SSI Serial Clock, not used in this application.
SSIWS	None, P205 Default: None	SSI Stereo pin selection, not used in this application.
SSIDATA	None, P206 Default: None	SSI Data pin selection, not used in this application.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.26.6 Using the I2S HAL Module in an Application

The typical steps in using the I2S HAL module in an application are:

1. Open the I2S HAL module using the i2s_api_t::open API.
2. Use the i2s_api_t::write API to write audio data to the I2S bus.
3. Wait for a callback with I2S_EVENT_TX_EMPTY and free the source buffer.
4. Use the i2s_api_t::read API to read data from the I2S bus.
5. Wait for a callback with I2S_EVENT_RX_FULL before accessing the destination buffer or reading the next buffer.
6. Use other APIs as needed by the application.
7. Close the module with the i2s_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

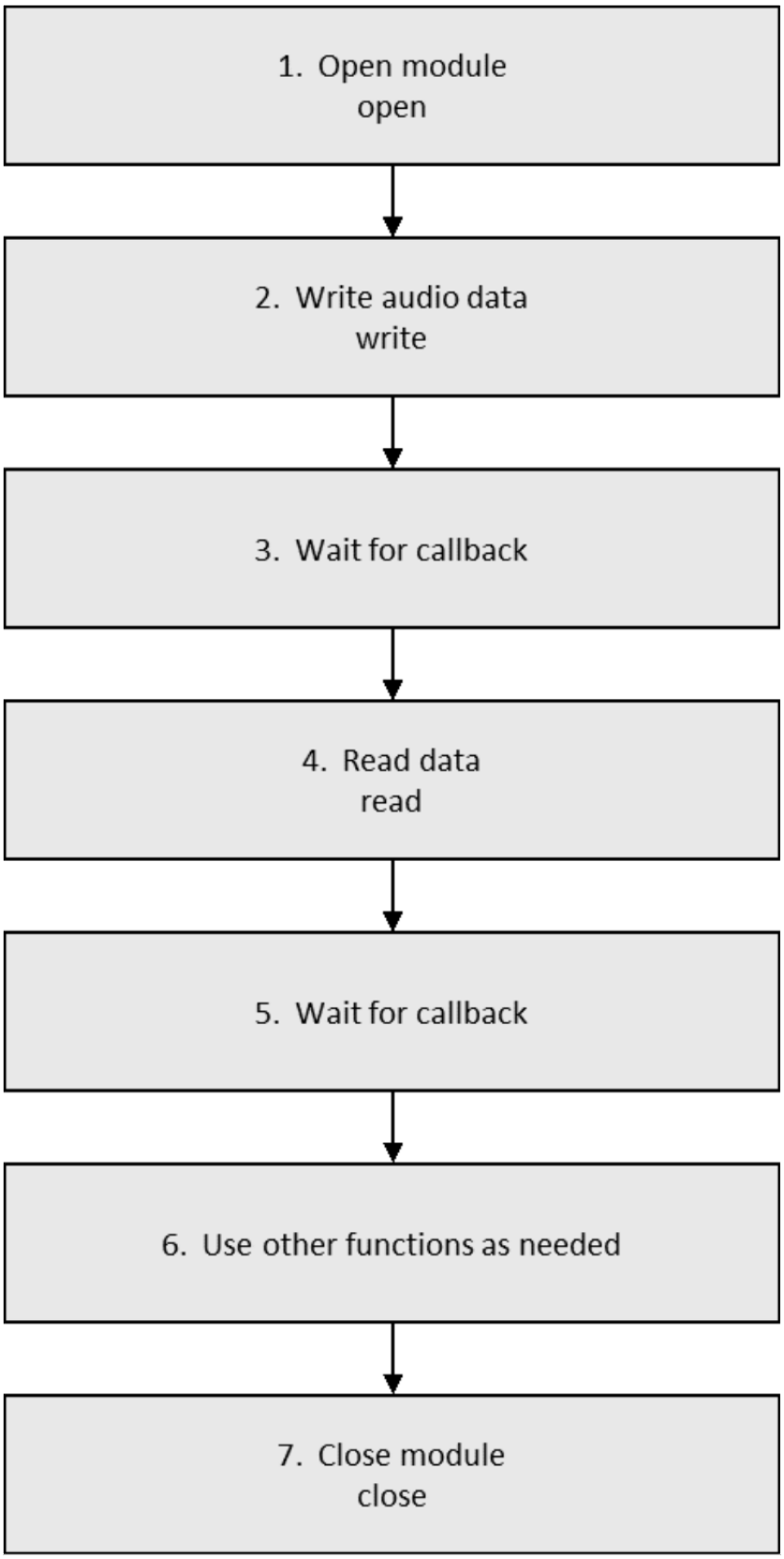


Figure 322: Flow Diagram of a Typical I2S HAL Module Application

4.2.27 GPT Input Capture on r_gpt Driver

4.2.27.1 GPT Input Capture HAL Module Introduction

The Input Capture HAL module provides an API for measuring input pulse-width and pulse-period measurement. The Input Capture HAL module also configures the input capture parameters to use with the GPT peripheral on the Synergy MCU. A user-defined callback can be created to acquire the value each time a new measurement is complete.

GPT Input Capture HAL Module Features

The Input Capture HAL module configures the GPT for an input capture function.

- The Input Capture HAL allows the user to perform the following tasks:
  - Initialize the module
  - Enable input capture measurement
  - Disable input capture measurement
  - Get the status (running or not) of the measurement counter
  - Get the last captured timer/overflows counter value
  - Close the input capture operation
- The Input Capture HAL module supports:
  - Pulse-width measurement and pulse-period measurement
  - Rising-edge or falling-edge measurement start
  - One-shot or periodic mode
  - Hardware-enable signals to enable captures (low enable/high enable)
  - Callback function with the following events:
    - Counter overflow
    - Input capture occur
  - Callback structure (input_capture_callback_args_t) that provides data on the interrupting event, including which interrupt occurs and the associated counter values.

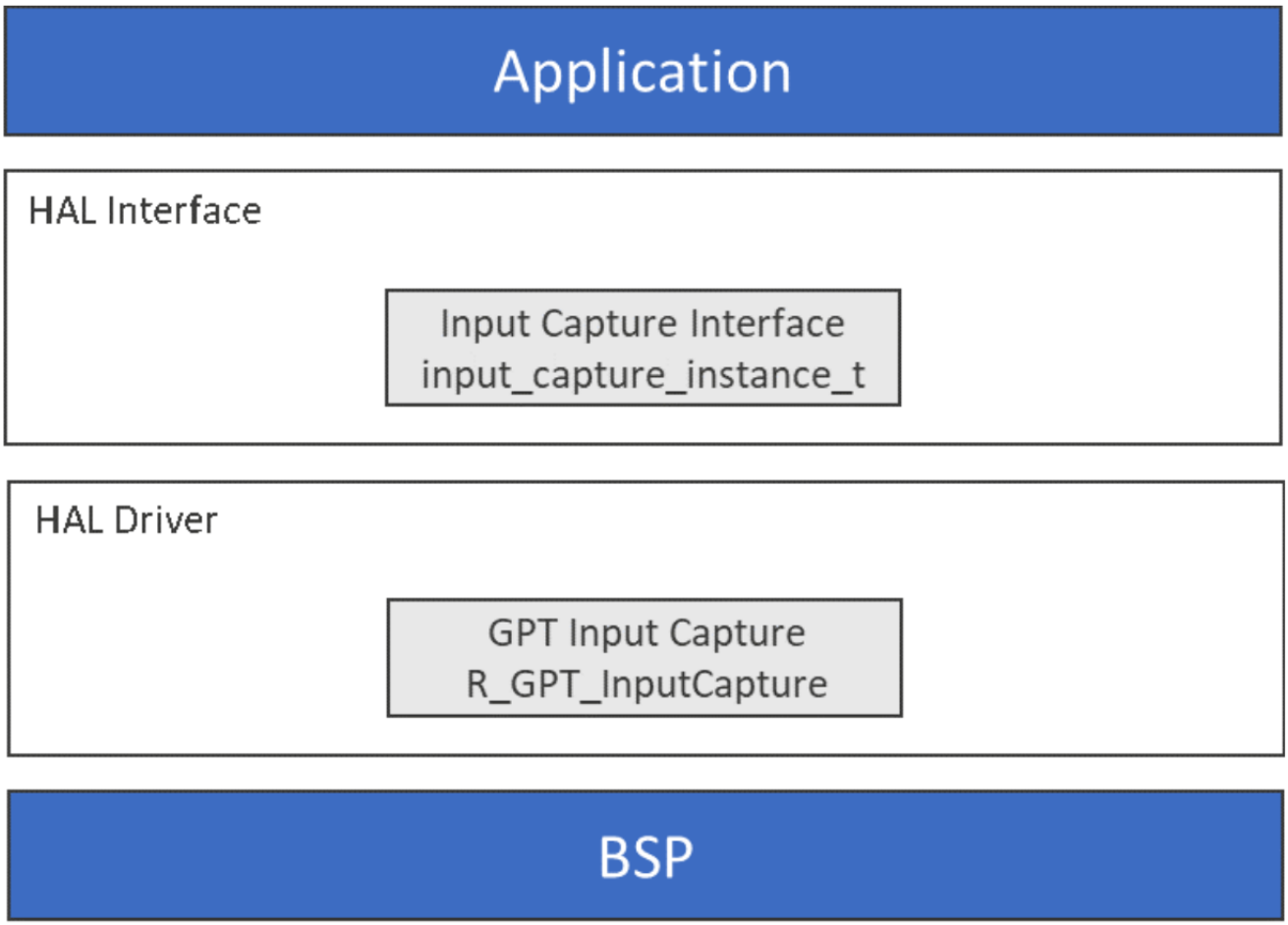


Figure 323: Input Capture HAL Module Block Diagram

GPT Input Capture Hardware support details

The following hardware features are, or are not, supported by SSP for GPT_INPUT_CAPTURE.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Saw Waves	Triangle Waves *	PWM waveform for controlling brushless DC motors	Compare match output for Low, High, and Toggle	Input capture function	Automatic addition of dead time
S124	☒	☒	☒	☒	✓	☒
S128	☒	☒	☒	☒	✓	☒

S1JA	☒	☒	☒	☒	✓	☒
S3A1	☒	☒	☒	☒	✓	☒
S3A3	☒	☒	☒	☒	✓	☒
S3A6	☒	☒	☒	☒	✓	☒
S3A7	☒	☒	☒	☒	✓	☒
S5D3	☒	☒	☒	☒	✓	☒
S5D5	☒	☒	☒	☒	✓	☒
S5D9	☒	☒	☒	☒	✓	☒
S7G2	☒	☒	☒	☒	✓	☒

MCU Group	PWM Mode	Phase Count Function	Event link function through ELC HAL driver	Noise filtering function
S124	☒	☒	☒	✓
S128	☒	☒	☒	✓
S1JA	☒	☒	☒	✓
S3A1	☒	☒	☒	✓
S3A3	☒	☒	☒	✓
S3A6	☒	☒	☒	✓
S3A7	☒	☒	☒	✓
S5D3	☒	☒	☒	✓
S5D5	☒	☒	☒	✓
S5D9	☒	☒	☒	✓
S7G2	☒	☒	☒	✓

4.2.27.2 GPT Input Capture HAL Module APIs Overview

The Input Capture HAL module interface defines APIs for opening, closing, enabling, disabling, accessing status information and last-capture value accessing using the General PWM Timer (GPT) with Input Capture. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the HAL Module API Summary.

Input Capture HAL Module API Summary

Function Name	Example API Call and Description
open	g_input_capture.p_api->open(g_input_capture.p_ctrl, g_input_capture.p_cfg); Opens the Input Capture HAL and initializes configuration.

close	g_input_capture.p_api->close(g_input_capture.p_ctrl); Closes the input capture operation. Allow drive to be reconfigured, and may reduce power consumption.
enable	g_input_capture.p_api->enable(g_input_capture.p_ctrl); Enables input capture measurement.
disable	g_input_capture.p_api->disable(g_input_capture.p_ctrl); Disables input capture measurement.
infoGet	g_input_capture.p_api->infoGet(g_input_capture.p_ctrl, &input_capture_info); Gets the status (running or not) of the measurement counter.
lastCaptureGet	g_input_capture.p_api->lastCaptureGet(g_input_capture.p_ctrl, &input_capture_counter); Gets the last captured timer/overflows counter value.
versionGet	g_input_capture.p_api->versionGet(&input_capture_version); Retrieve the API version with the input_capture_version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	One of the parameters is NULL. Or the channel requested in the p_cfg parameter may not be available on the device selected in r_bsp_cfg.h or p_cfg->mode is invalid.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value or ISR is not enabled.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.27.3 GPT Input Capture HAL Module Operational Overview

The Input Capture HAL module controls the GPT HAL module units on a Synergy microcontroller (as configured by the user). It directly accesses the GPT hardware without using any RTOS elements and provides convenient APIs to simplify development.

When a normal measurement is complete and a callback is available (with interrupts enabled,) the Input Capture HAL module invokes the callback with the argument input_capture_callback_args_t.

The argument input_capture_callback_args_t indicates the channel, the event input_capture_event_t, the value of the timer captured when the interrupt occurred, and the number of counter overflows that occurred during this measurement.

If the interrupts are not enabled, the values read by the APIs would be the last captured timer/overflows counter value.

GPT Input Capture HAL Module Important Operational Notes and Limitations

GPT Input Capture HAL Module Operational Notes

GPT Input Capture Measurement Mode

The input capture interface provides support for one-shot measurement and periodic measurement. The GPT hardware does not natively support one-shot functionality. Code is automatically included in the interrupt service routine (ISR) to stop and clear the timer. For this reason, ISRs must be enabled for one-shot mode, even when the callback is unused.

GPT Input Capture Signal

The input capture measurement starts when the input capture signal edge (rising or falling) is detected on the input capture signal pin (GTIOCA/GTIOCB) and the enable condition is met. The enable condition is defined by the enable level and can be disabled (none), or a specified low or high level on the input capture enable pin (GTIOCA/GTIOCB). The input capture enable pin is the pin not used as the input capture signal pin.

Converting Measurement Counts to Time

When a measurement completes, the raw-count data and the number of overflows is returned to the user in the callback function.

If desired, the raw measurement data can be converted to logical time units in the callback or user application. To convert the raw data, the current PCLKD clock frequency and its pre-scaler value, number of overflows, maximum counter value, and measurement counts should be considered. The measurement counts and the number of overflows are provided in the callback arguments input_capture_callback_args_t.

The recommended method to obtain the current PCLKD frequency is to use the cgc_api_t::systemClockFreqGet API. The input clock frequency is the PCLKD frequency divided by the pre-scalar value and is represented as clk_freq_hz in the following Input Capture Time Calculation table.

The maximum counter value on the S7G2 (all channels), S3A7 (all channels), and S124 (channel 0) is 0xFFFFFFFF. The maximum counter value for S124 (channels 1 - 6) is 0xFFFF. This maximum counter value plus one (since counter starts from zero) is represented as max_counts in the following table:

Input Capture Time Calculation

Desired Time Units	Formula
Nanoseconds (ns)	time_ns = ((overflows * max_counts) + counter) * 1000000000 / clk_freq_hz
Microseconds (us)	time_ns = ((overflows * max_counts) + counter) * 1000000 / clk_freq_hz
Milliseconds (ms)	time_ns = ((overflows * max_counts) + counter) * 1000 / clk_freq_hz
Seconds (s)	time_ns = ((overflows * max_counts) + counter) / clk_freq_hz

GPT Input Capture HAL Module Limitations

- Currently, the Input Capture HAL module supports only pulse-width measurement and pulse-period measurement.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.27.4 Including the GPT Input Capture HAL Module in an Application

This section describes how to include the Input Capture HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Input Capture Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Input Capture Driver is g_input_capture0. This name can be changed in the associated Properties window.)

Input Capture HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_input_captureInput Capture Driver on r_gpt_input_capture	**Threads**> **HAL/Common Stacks**	**New Stack**> **Driver**> **Timers** > **Input Capture Driver on r_gpt_input_capture**

When the Input Capture Driver on r_gpt_input_capture is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

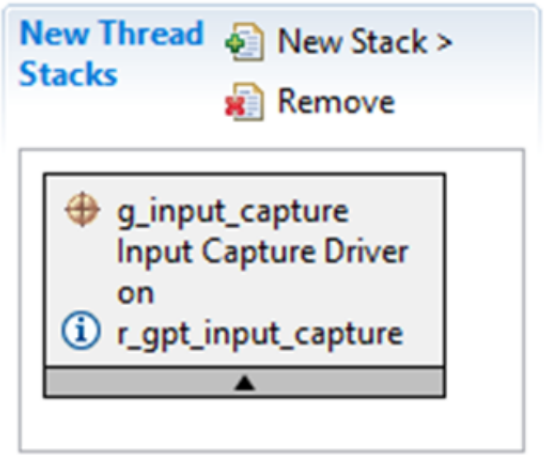


Figure 324: Input Capture HAL Module Stack

4.2.27.5 Configuring the GPT Input Capture HAL Module

The Input Capture HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Input Capture HAL Module on r_gpt_input_capture

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_input_capture	Module name.
Channel	0	Physical hardware channel.
Mode	Pulse Width, Period Default: Pulse Width	Measures inputs from the Signal edge until the opposite edge.
Signal Edge	Rising, Falling Default: Rising	Start measurement on rising or falling edge. Measurement stops on the opposite edge.

Repetition	Periodic, One Shot Default: Periodic	Capture a single measurement, then disable captures (one shot) until enable is called, or capture measurements continuously (periodic).
Auto Start	True, False Default: True	Set to true to enable measurements after configuring or false to leave the measurements disabled until enable is called.
Callback	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Input Capture Signal Pin	GTIOCA, GTIOCB Default: GTIOCA	Select the input pin used to trigger the start of a measurement.
GTIOCx Signal Filter	PCLK/1, PCLK/4, PCLK/16, PCLK/64 Default: PCLK/1	The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Clock Divider	PCLK/1, PCLK/4, PCLK/16, PCLK/64, PCKL/256, PCLK/1024 Default: PCLK/1	Clock divider used to scale the measurement counter.

Input Capture Enable Level	None, Low, High Default: None	Each GPT channel has 2 I/O pins (GTIOCA and GTIOCB). One must be selected as the Input Capture Signal Pin. The other GPT I/O pin can be used as a hardware enable signal to enable captures. Select None and captures are always enabled, select low and captures are enabled only while the enable input pin is low, select high and captures are enabled only while the enable input pin is high.
Input Capture Enable Filter	Enable, Disable Default: Disable (No filtering)	Enable/Disable the input noise filter for input on GTIOCx pin
Capture Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Capture interrupt priority selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Overflow interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

GPT Input Capture HAL Module Clock Configuration

The GPT HAL module uses the PCLKD as its clock source. The PCLKD frequency is set using the SSP configurator clocks tab prior to a build, or using the CGC Interface at run-time.

GPT Input Capture HAL Module Pin Configuration

To access a particular channel and pin, the GTIOCx pins must be set in the **Pins** tab of the ISDE. The following table provides the method for selecting the pins within the SSP configuration window and the subsequent table provides an example selection for GTIOCx pins.

Pin Selection for the Input Capture HAL Module on r_gpt_input_capture

Resource	ISDE Tab	Pin selection Sequence
GPT Input Capture	Pins	Select **Peripherals**> **Timer: GPT**> **GPT0**

Note

The selection sequence assumes GPT0 is the desired hardware target for the driver.

Pin Configuration Settings for the Input Capture HAL Module on r_kint

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only Default: Mixed	Pin grouping selection.
Operation Mode	Disabled, GTIOCA or GTIOCB, GTIOCA and GTIOCB Default: Disable	Select GTIOCA or GTIOCB as the Operation Mode for Input Capture on GPT.
GTIOCA	None, P300, P512 Default: None	GTIOCA Pin.
GTIOCB	None, P108, P511 Default: None	GTIOCB Pin.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.27.6 Using the GPT Input Capture HAL Module in an Application

The typical steps in using the Input Capture HAL module in an application are:

1. Initialize the module using the input_capture_api_t::open API.
2. The desired value can be found either in the main loop routine using the input_capture_api_t::lastCaptureGet API or in the callback function.
3. The capture and overflow interrupt can be disabled and the timer stopped using theinput_capture_api_t::disable API.
4. The capture and overflow interrupt can be enabled and the timer started using the input_capture_api_t::enable API.
5. The status of the captured counter (running or stopped) can be queried using input_capture_api_t::infoGet API.
6. The module can be closed using the input_capture_api_t::close API once done.

These common steps are illustrated in a typical operational flow diagram in the following figure:

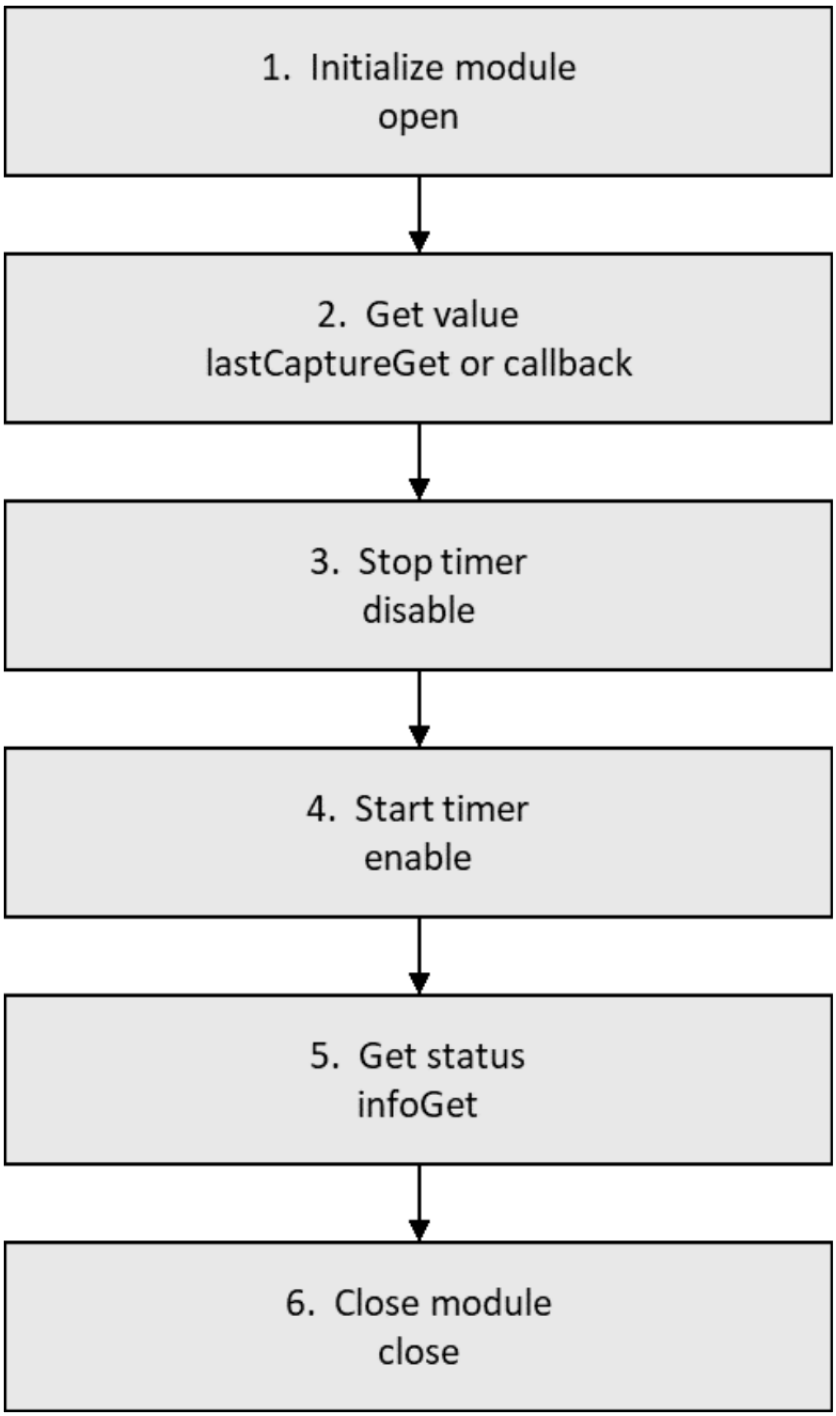


Figure 325: Flow Diagram of a Typical Input Capture HAL Module Application

4.2.28 I/O Port Driver

4.2.28.1 I/O PORT HAL Module Introduction

The I/O Port HAL module implements a high-level API for controlling I/O pins, configuring the board's pins and manipulating I/O pins. The operating state of an I/O pin can be set via the Synergy configurator. When the Synergy project is built, a pin configuration file is created. When the application runs, the BSP will configure the MCU IO port accordingly, using the same API functions described in this document.

I/O PORT HAL Module Features

The I/O PORT HAL module can perform the following functions:

- Create an event link between two blocks.
- Break that event link between two blocks.
- Generate one of two software events that interrupt the CPU.

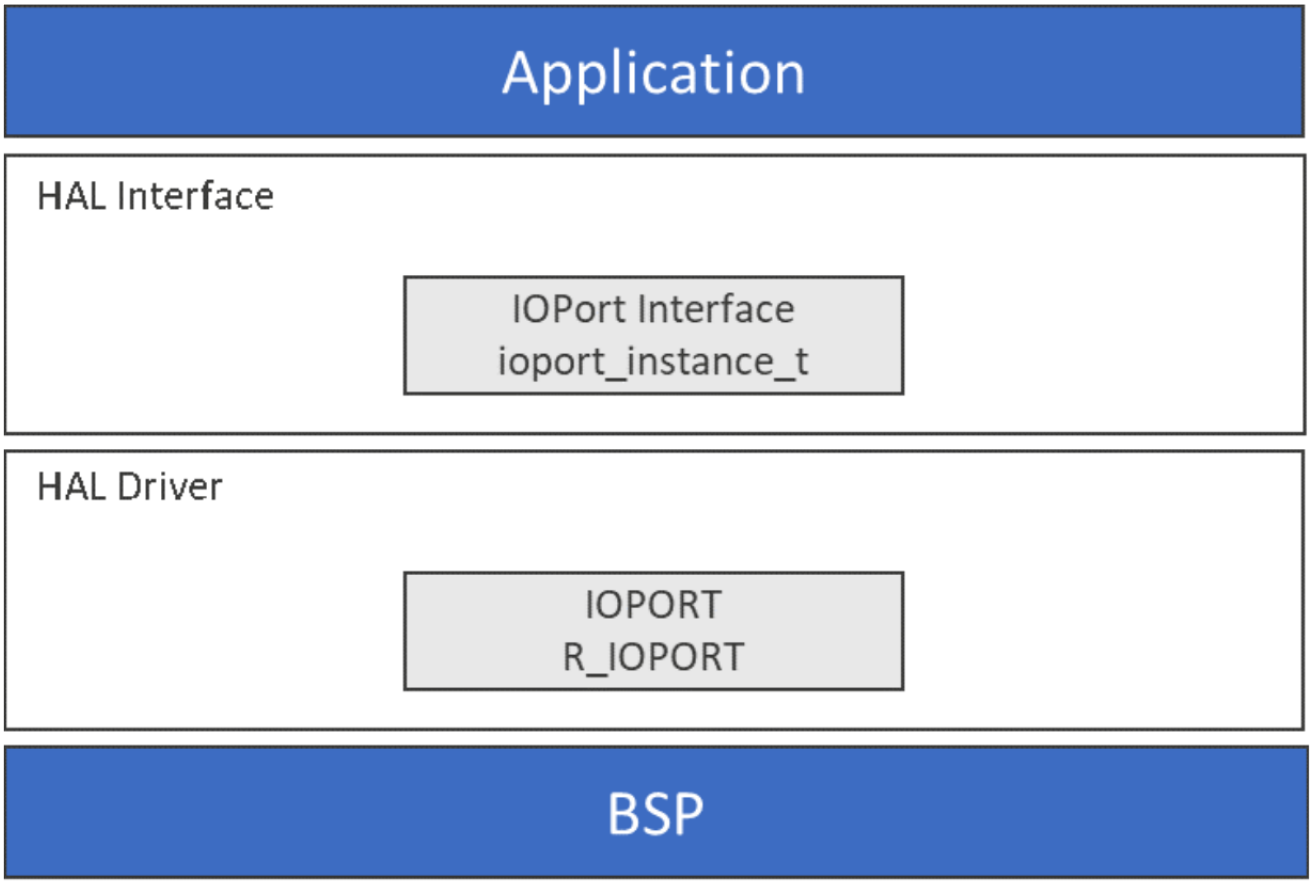


Figure 326: I/O PORT HAL Module Block Diagram

I/O Port Hardware support details

The following hardware features are, or are not, supported by SSP for GPIO.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A				Not supported by MCU			
MCU Group	Port Direction Setting	Input Data Read function	Output Port Write function	Pin Mode Control	Ethernet Mode Configuration	ELC_PORTn Event Input Read function*	ELC_PORTn Event Output Setting *
S124	✓	✓	✓	✓	N/A	✓	✓
S128	✓	✓	✓	✓	N/A	✓	✓
S1JA	✓	✓	✓	✓	N/A	✓	✓
S3A1	✓	✓	✓	✓	N/A	✓	✓
S3A3	✓	✓	✓	✓	N/A	✓	✓
S3A6	✓	✓	✓	✓	N/A	✓	✓
S3A7	✓	✓	✓	✓	N/A	✓	✓
S5D3	✓	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓	✓

- Note: Event Linking would have to be set up by the user, rather than using the ELC API.

4.2.28.2 I/O PORT HAL Module APIs Overview

The I/O Port HAL module defines APIs for reading and writing from particular pins and ports. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I/O PORT HAL Module API Summary

Function Name	Example API Call and Description
init	g_ioport.p_api->init(g_ioport.p_cfg); Initialize configuration of multiple pins.
pinCfg	g_ioport.p_api->pinCfg(IOPORT_PORT_00_PIN_00, IOPORT_CFG_IRQ_ENABLE | IOPORT_CFG_PORT_DIRECTION_INPUT); Configure settings for an individual pin.
pinDirectionSet	g_ioport.p_api->pinDirectionSet(IOPORT_PORT_00_PIN_00, IOPORT_DIRECTION_INPUT); Set the pin direction of a pin.
pinEventInputRead	g_ioport.p_api->pinEventInputRead(IOPORT_PORT_00_PIN_00, &pin_level); Read the event (ELC) input data of the specified pin and return the level.

pinEventOutputWrite	g_ioport.p_api->pinEventOutputWrite(IOPORT_PORT_00_PIN_00, IOPORT_PIN_LEVEL_HIGH); Write pin event (ELC) data.
pinEthernetModeCfg	g_ioport.p_api->pinEthernetModeCfg(IOPORT_ETHERNET_CHANNEL_0, IOPORT_ETHERNET_MODE_MII); Configure the PHY mode of the Ethernet channels.
pinRead	g_ioport.p_api->pinRead(IOPORT_PORT_00_PIN_00, &pin_level); Read level of a pin.
pinWrite	g_ioport.p_api->pinWrite(IOPORT_PORT_00_PIN_00, IOPORT_PIN_LEVEL_HIGH); Write specified level to a pin.
portDirectionSet	g_ioport.p_api->portDirectionSet(IOPORT_PORT_00, direction_values, mask); Set the direction of one or more pins on a port.
portEventInputRead	g_ioport.p_api->portEventInputRead(IOPORT_PORT_00, &pin_levels); Read captured event (ELC) data for a port.
portEventOutputWrite	g_ioport.p_api->portEventOutputWrite(IOPORT_PORT_00, pin_levels, mask); Write event (ELC) output data for a port.
portRead	g_ioport.p_api->portRead(IOPORT_PORT_00, &pin_levels); Read states of pins on the specified port.
portWrite	g_ioport.p_api->portWrite(IOPORT_PORT_00, pin_levels, mask); Write to multiple pins on a port.
versionGet	g_ioport.p_api->versionGet(&version); Retrieve version information using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	The port/pin/mask/direction/level (and so forth) not valid.
SSP_ERR_ASSERTION	Unexpected null pointer.

SSP_ERR_UNSUPPORTED	Feature not supported – for instance Ethernet configuration not supported on the device.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.28.3 I/O PORT HAL Module Operational Overview

The I/O Port HAL module provides the ability to access the I/O ports of a device at both bit and port level; both port and pin direction can be changed. In addition, a number of configuration APIs are provided to change the functionality of individual pins.

The I/O Port HAL module provides the following operations for configuring pins:

- Initializes the driver – performed by calling ioport_api_t::init API:
  - Performs parameter checking and processes error conditions.
  - Handles VBATT domain pin configuration.
  - Writes PFS registers for pins.
- Configures pin – performed by calling ioport_api_t::pinCfg API:
  - Performs parameter checking and processes error conditions (checks pin number pin, VBATT support).
  - Writes PFS register for the pin.
- Reads pin level – performed by calling ioport_api_t::pinRead API:
  - Performs parameter checking and processes error conditions (checks pin number pin).
  - Reads PFS register for the pin.
- Reads all pin levels on a port – performed by calling ioport_api_t::portRead API:
  - Performs parameter checking and processes error conditions (checks port number port).
  - Reads current value of PCNTR register value for the specified port.
- Writes pin level – performed by calling ioport_api_t::pinWrite API:
  - Performs parameter checking and processes error conditions (check pin number pin and written level level).
  - Write to PFS register for the pin.
- Write multiple pin levels on a port – performed by calling ioport_api_t::portWrite API:
  - Performs parameter checking and processes error conditions (checks port number port and pin mask mask).
  - Reads current configuration from the PCNTR register for the specified port.
  - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask, preserving pin levels out of the scope.
- Sets the direction of multiple pins on a port – performed by calling ioport_api_t::portDirectionSet API:
  - Performs parameter checking and processes error conditions (checks port number port and pin mask mask).
  - Reads current configuration from the PCNTR register for the specified port.
  - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask, preserving pin directions out of the scope.
- Writes pin direction – performed by calling ioport_api_t::pinDirectionSet API:
  - Performs parameter checking and processes error conditions (checks pin number pin and written direction direction).
  - Writes to the PFS register for the pin.
- Reads event (ELC) port input – performed by calling ioport_api_t::portEventInputRead API:
  - Performs parameter checking and processes error conditions (checks port number

port).
    - Reads current value of the PCNTR register value for the specified port.
- Reads event (ELC) pin input – performed by calling ioport_api_t::pinEventInputRead API:
    - Performs parameter checking and processes error conditions (checks pin number pin).
    - Reads current value of the PCNTR register value for the specified pin's port.
    - Gets the pin level by applying a pin mask to the PCNTR register value.
- Writes event (ELC) port output – performed by calling the ioport_api_t::portEventOutputWrite API:
    - Performs parameter checking and processes error conditions (checks port number port and pin mask mask_value).
    - Reads current configuration from the PCNTR register for the specified port.
    - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask preserving pin levels out of the event scope.
- Writes event (ELC) pin output – performed by calling ioport_api_t::pinEventOutputWrite API:
    - Performs parameter checking and processes error conditions (checks pin number pin and written level ioport_api_t::pinRead).
    - Writes the pin level to the PCNTR register for the specified pin's port accordingly to the mask that is preserving pin levels out of the event scope.
- Configures Ethernet channel PHY mode – performed by calling ioport_api_t::pinEthernetModeCfg API:
    - Performs parameter checking and processes error conditions (checks Ethernet channel channel and mode mode).
    - Updates the Ethernet Control Register (PFENET).

I/O PORT HAL Module Important Operational Notes and Limitations

I/O PORT HAL Module Operational Notes

- A bit mask of 16 bits needs to be applied in order to read and write to a specific pin on a port; ports are numbered from 0 (LSB) to 15 (MSB).
- Configuring the Ethernet port (selection of the RMII or MII output format) using the ioport_api_t::pinEthernetModeCfg API function on MCUs without an Ethernet Port will return an Unsupported Error message. To avoid this, the developer should verify the target MCU has an Ethernet port available. This information is easily found in the target MCU hardware user's manual. Simply verify that an Ethernet peripheral is available on the target MCU before beginning development.

I/O PORT HAL Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.28.4 Including the I/O PORT HAL Module in an Application

This section describes how to include the I/O PORT HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I/O Port Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I/O Port Driver is g_ioport0. This

name can be changed in the associated Properties window.)

I/O PORT HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ioport I/O Port driver on r_ioport	Threads	**Highlight HAL/Common and select New> Driver> System> I/O Port Driver on r_ioport**

When the I/O Port Driver on r_ioport is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

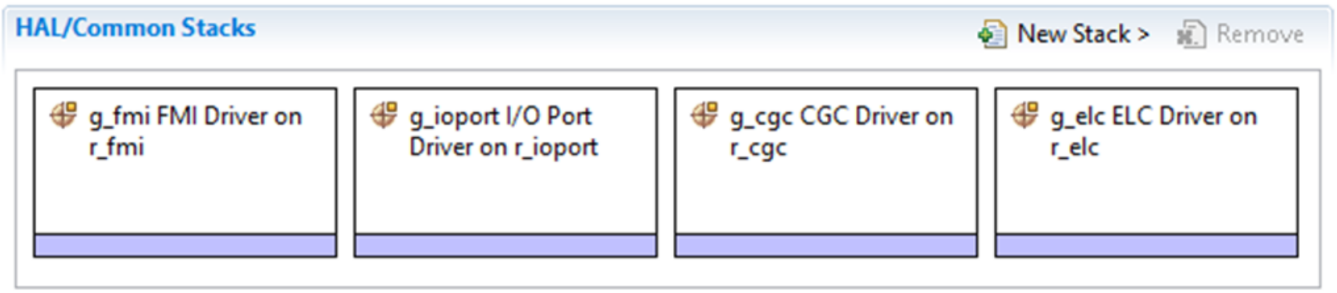


Figure 327: I/O PORT HAL Module Stack

4.2.28.5 Configuring the I/O PORT HAL Module

The I/O PORT HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I/O PORT HAL Module on r_ioport

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_ioport	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

I/O PORT HAL Module Clock Configuration

The I/O PORT HAL module does not require a specific clock configuration.

I/O PORT HAL Module Pin Configuration

There are no pins associated directly with the I/O PORT HAL Module that require configuration.

4.2.28.6 Using the I/O PORT HAL Module in an Application

The typical steps in using the I/O PORT HAL module in an application are:

1. Initialize the driver using the ioport_api_t::init API.
2. Configure the pins using the ioport_api_t::pinCfg API.
3. Read from specified pins and ports using the ioport_api_t::pinRead and ioport_api_t::portRead APIs.
4. Write to specified pins and ports using the ioport_api_t::pinWrite and ioport_api_t::portWrite APIs.

These common steps are illustrated in a typical operational flow diagram in the following figure:

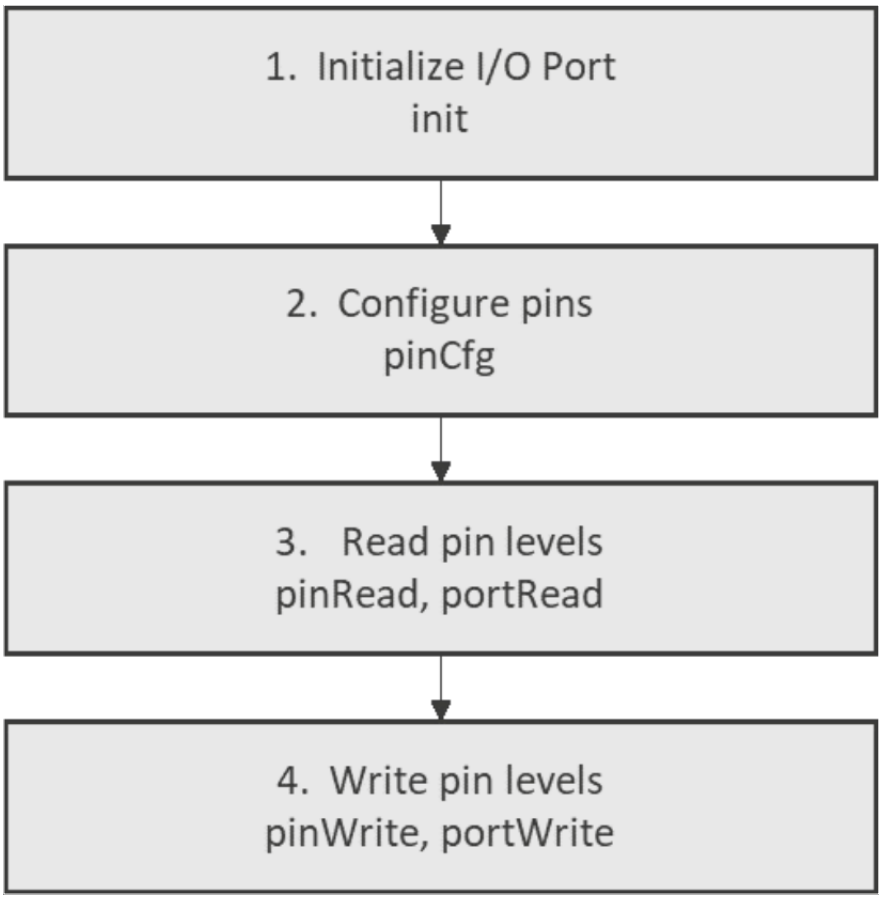


Figure 328: Flow Diagram of a Typical I/O PORT HAL Module Application

4.2.29 Watchdog Driver on r_iwdt

4.2.29.1 Independent Watchdog Timer HAL Module Introduction

The Independent Watchdog Timer (IWDT) HAL module provides a high-level API for watchdog timer applications and uses the IWDT peripheral on the Synergy MCU. A user-defined callback can be created to respond to event notifications.

Independent Watchdog Timer HAL Module Features

The IWDT HAL module supports the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an NMI
- Supports the internal Watchdog timer peripheral (IWDT), which has its own clock source which improves safety.
- Supports automatic hardware configuration after reset.

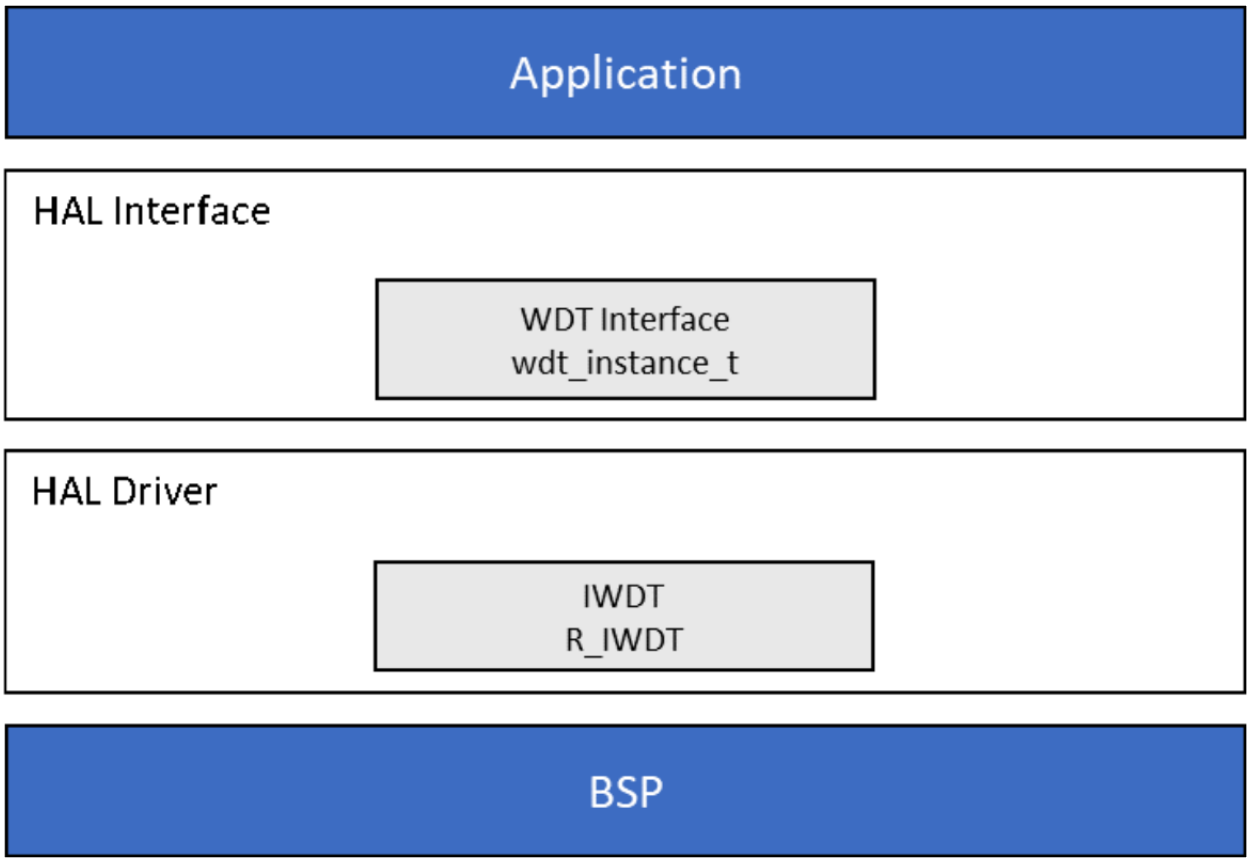


Figure 329: Independent Watchdog Timer HAL Module Block Diagram

Independent Watchdog Timer Hardware Support Details

The following hardware features are, or are not, supported by SSP for IWDT:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Count down	Autostart mode	Reset output	Interrupt request output
S124	✓	✓	✓	✓
S128	✓	✓	✓	✓
S1JA	✓	✓	✓	✓
S3A1	✓	✓	✓	✓
S3A3	✓	✓	✓	✓
S3A6	✓	✓	✓	✓

S3A7	✓	✓	✓	✓
S5D3	✓	✓	✓	✓
S5D5	✓	✓	✓	✓
S5D9	✓	✓	✓	✓
S7G2	✓	✓	✓	✓

MCU Group	Sleep mode count stop control output	Event link function through ELC HAL driver	Window function	Conditions for stopping the Counter – reset/underflow- refresh error	Refresh error and under flow error detect	Reading the counter value
S124	☒	☒	✓	✓	✓	✓
S128	☒	☒	✓	✓	✓	✓
S1JA	☒	☒	✓	✓	✓	✓
S3A1	☒	☒	✓	✓	✓	✓
S3A3	☒	☒	✓	✓	✓	✓
S3A6	☒	☒	✓	✓	✓	✓
S3A7	☒	☒	✓	✓	✓	✓
S5D3	☒	☒	✓	✓	✓	✓
S5D5	☒	☒	✓	✓	✓	✓
S5D9	☒	☒	✓	✓	✓	✓
S7G2	☒	☒	✓	✓	✓	✓

MCU Group	Selecting the clock frequency division ratio after a reset	Selecting the timeout period of the independent watchdog timer
S124	☒	✓ Set by the OFS registers in the BSP
S128	☒	✓ Set by the OFS registers in the BSP
S1JA	☒	✓ Set by the OFS registers in the BSP
S3A1	☒	✓ Set by the OFS registers in the BSP
S3A3	☒	✓ Set by the OFS registers in the BSP
S3A6	☒	✓ Set by the OFS registers in the BSP

S3A7	☒	✓ Set by the OFS registers in the BSP
S5D3	☒	✓ Set by the OFS registers in the BSP
S5D5	☒	✓ Set by the OFS registers in the BSP
S5D9	☒	✓ Set by the OFS registers in the BSP
S7G2	☒	✓ Set by the OFS registers in the BSP

4.2.29.2 Independent Watchdog Timer HAL Module APIs Overview

The IWDT HAL module defines APIs for open, refresh, read and get status. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Independent Watchdog Timer HAL Module API Summary

Function Name	Example API Call and Description
cfgGet	g_wdt0.p_api->cfgGet(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the iWDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
open	g_wdt0.p_api->open(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the iWDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
refresh	g_wdt0.p_api->refresh(g_wdt0.p_ctrl); Refresh the watchdog timer.
statusGet	g_wdt0.p_api->statusGet(g_wdt0.p_ctrl, &status); Read the status of the iWDT.
statusClear	g_wdt0.p_api->statusClear(g_wdt0.p_ctrl, clear); Clear the status flags of the iWDT.
counterGet	g_wdt0.p_api->counterGet(g_wdt0.p_ctrl, &counter); Read the current iWDT counter value.
tiimeoutGet	g_wdt0.p_api->timeoutGet(g_wdt0.p_ctrl, &timeout); Read the watchdog timeout values.

versionGet	g_wdt0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully executed.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.
SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.29.3 Independent Watchdog Timer HAL Module Operational Overview

The IWDT HAL module configures the IWDT Interface. When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:

- Resetting of the device
- Generation of an NMI

The following figure shows an example of the operation of the IWDT. When refreshed in the valid refresh period of the counter the timer count value is reset. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device or generates an NMI.

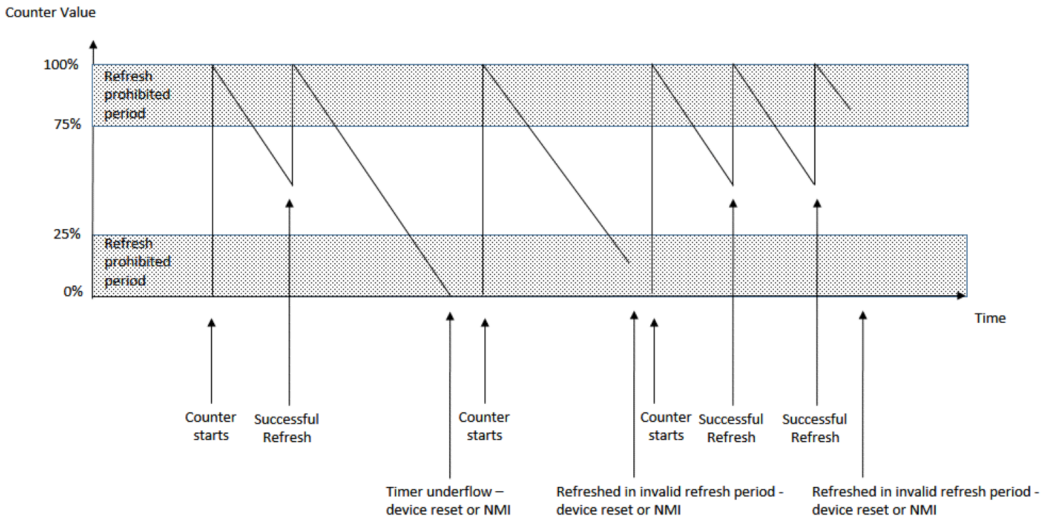


Figure 330: Independent Watchdog Timer HAL Module Operational Diagram

All series of Synergy microcontrollers have an option-setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

The following table details which parameters of the IWDT can be configured by the OFS registers.

Note

The IWDT can only be configured via the OFS registers. The IWDT does not support Register Start mode.

Control	Description
IWDT Start Mode Select	Automatically starts the IWDT after a Reset, if enabled.
IWDT Timeout Period	Specifies the IWDT timeout (number of clock cycles) 128 cycles 512 cycles 1024 cycles 2048 cycles
IWDT-Dedicated Clock Frequency Division Ratio	1 1/16 1/32 1/64 1/128 1/256

IWDT Window End Position	25 %50 %75 %100% (no window end position set)
IWDT Window Start Position	25 %50 %75 %100% (no window start position set)
IWDT Reset Interrupt Request	The IWDT can either generate an Interrupt Signal or a Reset signal.
IWDT Stop Control	The IWDT can continue to count or Stop counting in Low Power Mode.

Note

For further information on the contents of the OFS0 register, see the Synergy MCU hardware manual.

The OFS register values are set via the properties dialog of the **BSP** tab of Synergy Configuration editor as shown in the following figures:

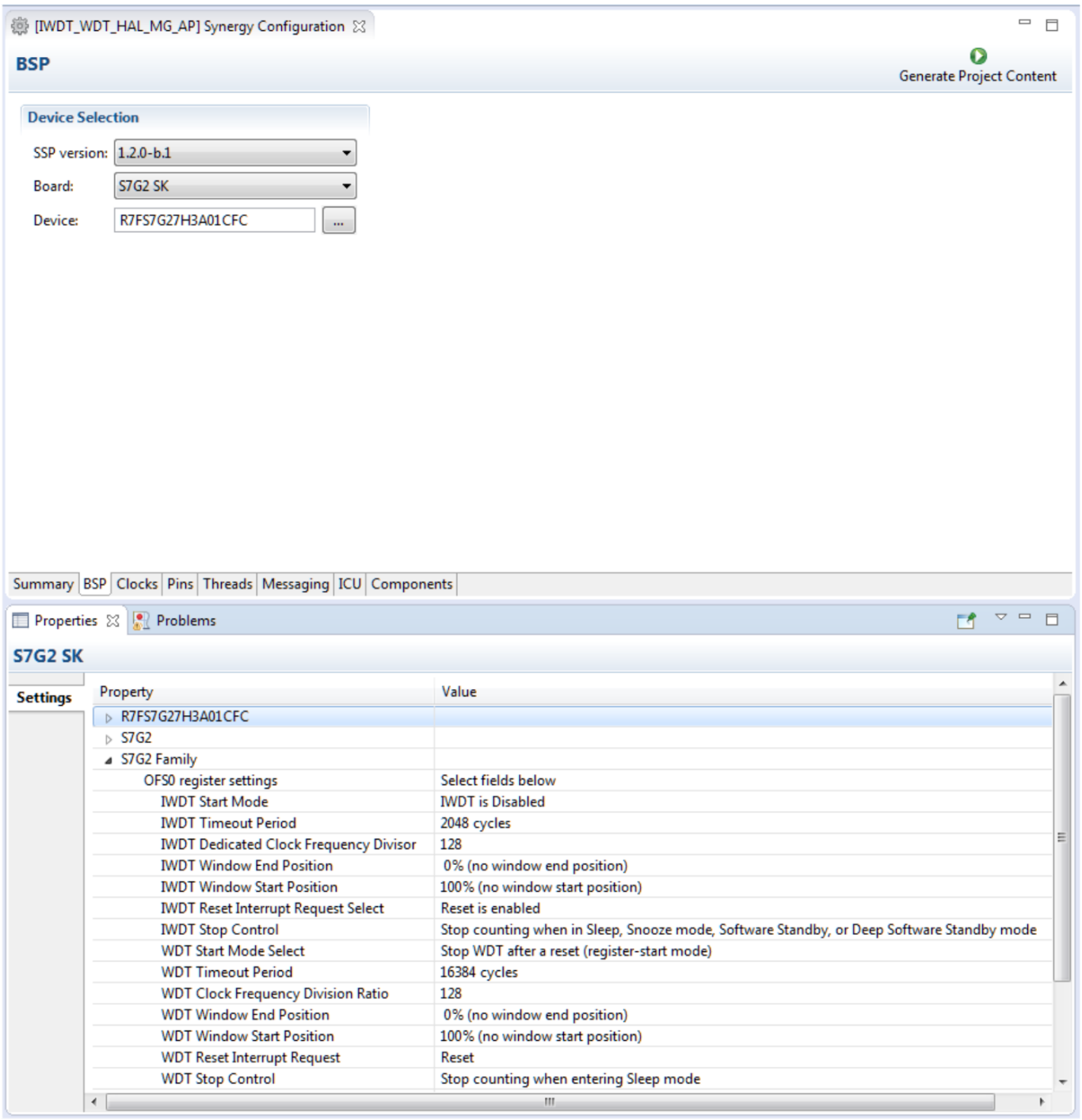


Figure 331: Independent Watchdog Timer HAL Module Configuration Screens

Independent Watchdog Timer HAL Module Important Operational Notes and Limitations

Independent Watchdog Timer HAL Module Operational Notes

IWDT HAL Module Period Calculation

The IWDT operates from IWDTCLK. Assuming largest parameters for the WDT and an IWDTCLK frequency of 15 kHz, the time from the last refresh to device reset or NMI generation will be just under 35 seconds as detailed below.

IWDTLCK = 15 kHz

Clock division ratio = IWTCLK/256

Timeout period = 2048 cycles

IWDT clock frequency = 15 kHz / 256 = 58.59 Hz

Cycle time = 1 / 58.59 Hz = 17.07 ms

Timeout = 17.07 ms x 2048 cycles = 34.95 seconds

Triggering DMAC/DTC with the IWDT HAL Module

To trigger a transfer of data using the DMAC or DTC peripheral when the watchdog counter underflows or when a refresh is attempted outside of the valid refresh period, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_IWDT_UNDERFLOW. See the appropriate module guide for additional information.

Triggering ELC Events with the IWDT HAL Module

The IWDT can trigger the start of other peripherals as available with the Event Link Controller (ELC). See the ELC HAL module guide for additional information.

Independent Watchdog Timer HAL Module Limitations

- When using a J-Link debugger, the IWDT counter does not count and therefore will not reset the device or generate an NMI.
- When there is no active task ready to run, ThreadX puts the MCU into sleep mode. If the IWDT is configured to stop the counter in low-power mode, then your application must restart the timer when used with the ThreadX RTOS.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.29.4 Including the Independent Watchdog Timer HAL Module in an Application

This section describes how to include the Independent Watchdog Timer HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Independent Watchdog Timer Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Independent Watchdog Timer Driver is g_iwdt0. This name can be changed in the associated Properties window.)

Independent Watchdog Timer HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_wdt0 IWDT HAL on r_iwdt	Threads	**New Stack> Driver> Monitoring> IWDT HAL on r_iwdt**

When the Independent Watchdog Timer Driver on r_iwdt is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

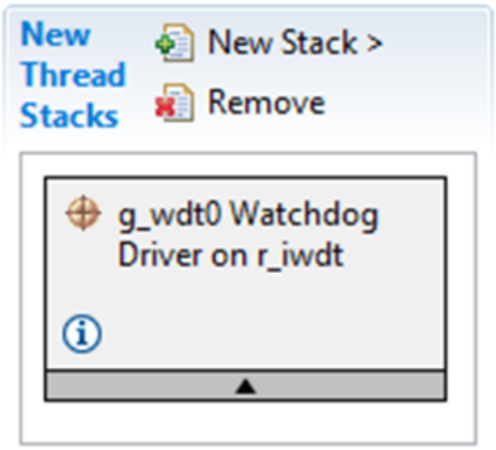


Figure 332: Independent Watchdog Timer HAL Module Stack

4.2.29.5 Configuring the Independent Watchdog Timer HAL Module

The Independent Watchdog Timer HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Independent Watchdog Timer HAL Module on r_iwdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_wdt0	Module name.

NMI Callback	NULL	Callback. A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configure Option Function Select Register 0 (OFS0) for the IWDT HAL Module

All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

Configure the Interrupts for the IWDT HAL Module

Use the ISDE to configure the IWDT interrupts in the same way as configuring the other options for the IWDT module. If the IWDT is configured to generate an NMI interrupt on underflow or invalid refresh the interrupt must be enabled in the BSP.

Note

To enable interrupts, set the priority of the IWDT > IWDT NMIUNDF N. This sets BSP_IRQ_CFG_IWDT_UNDERFLOW in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the IWDT NMIUNDF N interrupt is enabled in the BSP, the corresponding ISR will be defined. The ISR will call a user callback function if one was registered in the wdt_api_t::open API.

Independent Watchdog Timer HAL Module Clock Configuration

The IWDT has its own dedicated clock operating at a set frequency which cannot be modified.

Independent Watchdog Timer HAL Module Pin Configuration

The IWDT does not require pins for its operation.

4.2.29.6 Using the Independent Watchdog Timer HAL Module in an Application

The typical steps in using the Independent Watchdog Timer HAL module in an application are:

1. Register the IWDT NMI callback using the wdt_api_t::open API.
2. Read the configuration of the IWDT HAL module using the wdt_api_t::cfgGet API.
3. Refresh the independent watchdog timer using the wdt_api_t::refresh API.

4. Read the IWDT status flags using the wdt_api_t::statusGet API.
5. Clear the IWDT Status and error flags using the wdt_api_t::statusClear API.
6. Read the current count value of the IWDT using the wdt_api_t::counterGet API.
7. Read the timeout values of the IWDT HAL module using the wdt_api_t::timeoutGet API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

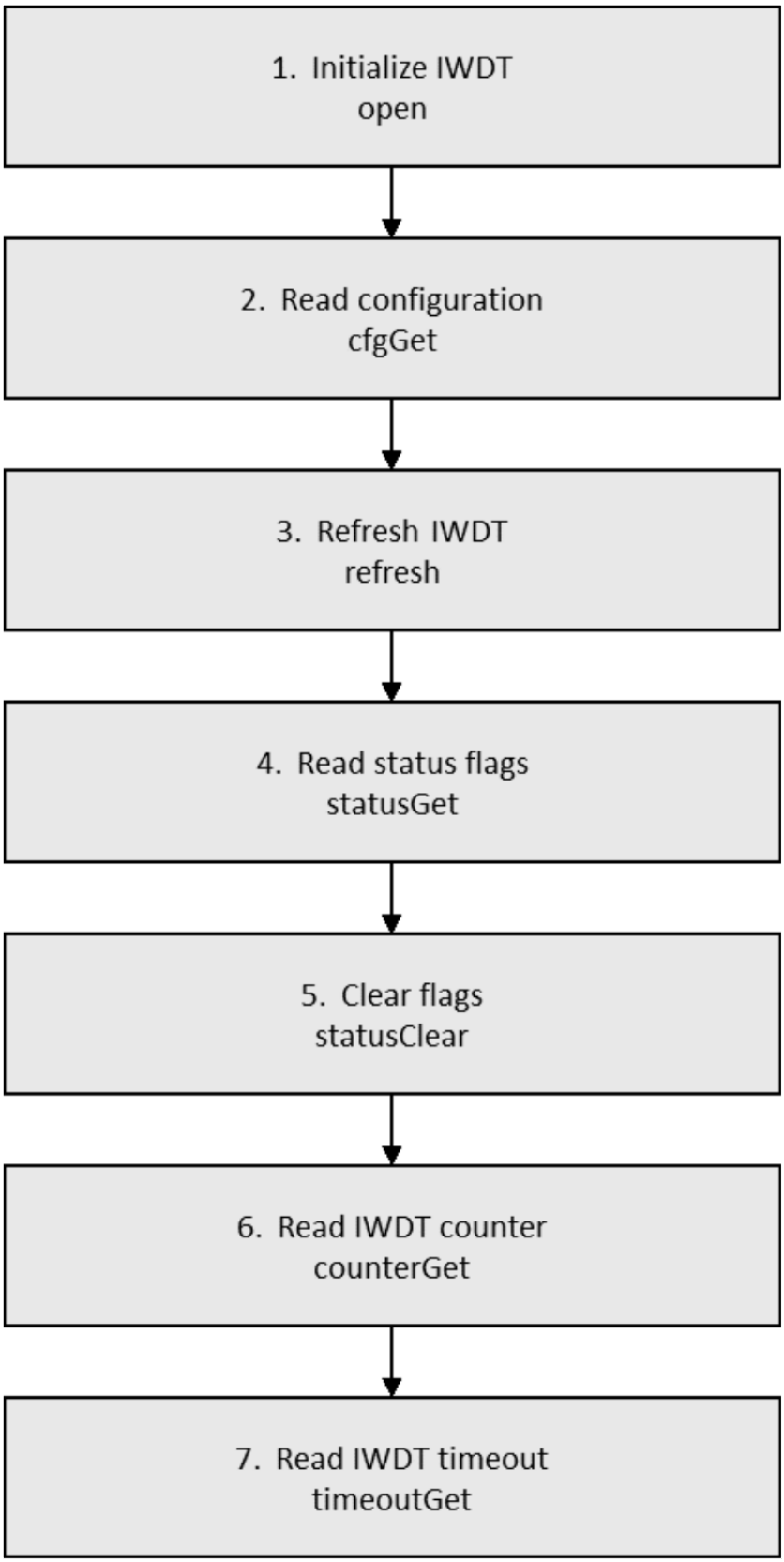


Figure 333: Flow Diagram of a Typical Independent Watchdog Timer HAL Module Application

4.2.30 JPEG Decode Driver

4.2.30.1 JPEG Decode HAL Module Introduction

The JPEG Decode HAL module provides high-level APIs for JPEG Decode image processing. The JPEG Decode HAL module uses the Synergy MCU JPEG Codec peripheral. A user callback function is available to inform the application program of key processing events.

JPEG Decode HAL Module Features

- Supports JPEG decompression.
- Supports polling mode that allows an application to wait for JPEG Decoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into JPEG Decoder module. This feature allows an application to read coded JPEG image from a file or from network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded format YCbCr444, YCbCr422, YCbCr420, YCbCr411.
- Supports the output format ARGB8888, RGB565.
- Returns error when the JPEG image's size, height and width do not meet the requirements.

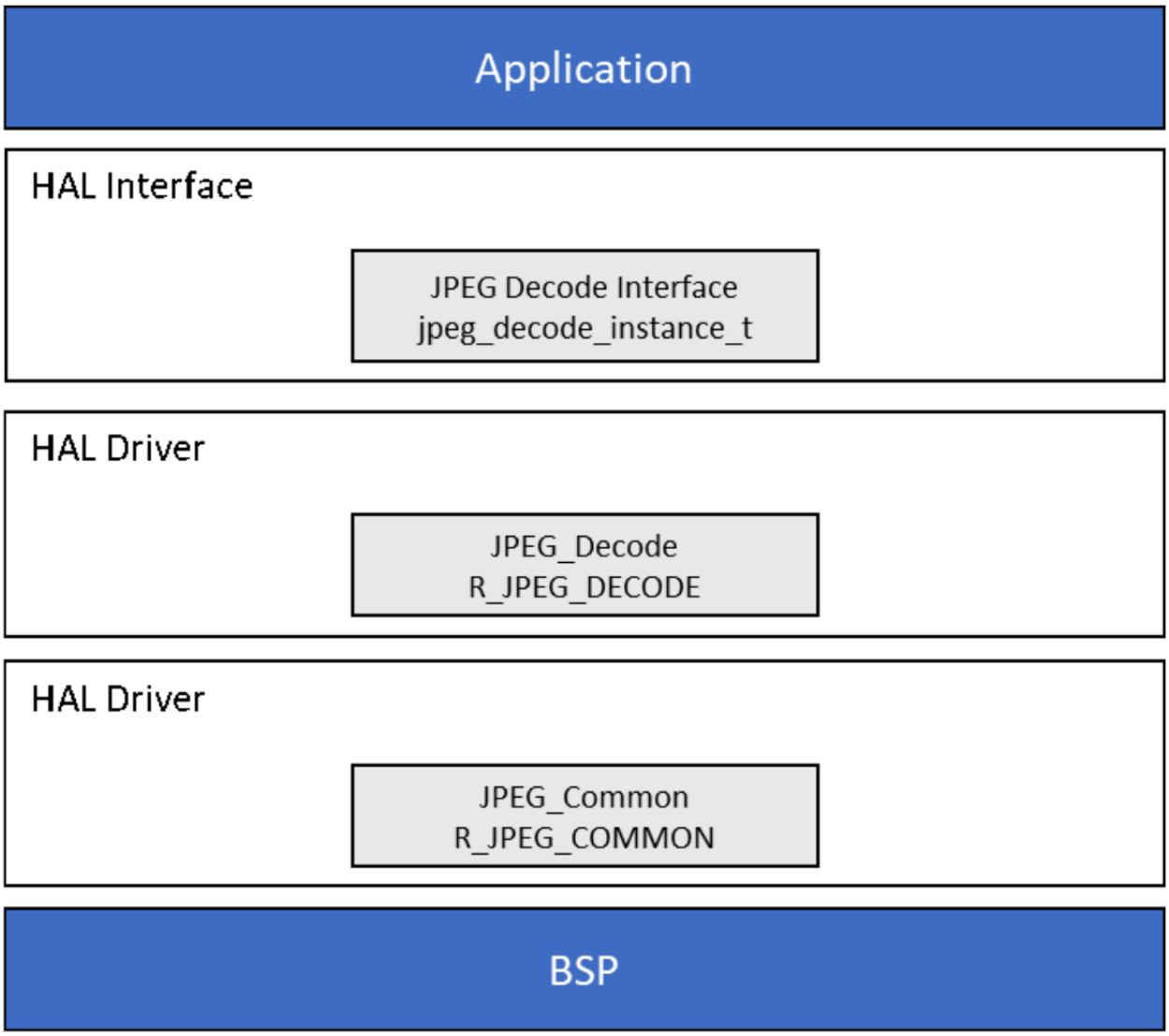


Figure 334: JPEG Decode HAL Module Block Diagram

JPEG Decode Hardware support details

The following hardware features are, or are not, supported by SSP for JPEG.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	8 lines by 8 pixels in YCbCr444	8 lines by 16 pixels in YCbCr422	8 lines by 32 pixels in YCbCr411	16 lines by 16 pixels in YCbCr420	Output decoded format ARGB8888	Output decoded format RGB565
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

4.2.30.2 JPEG Decode HAL Module APIs Overview

The JPEG Decode HAL Module implements APIs to open, set processing parameters, start processing, get processing status and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

JPEG Decode HAL Module API Summary

Function Name	Example API Call and Description
open	g_jpeg_decode0.p_api->open(g_jpeg_decode0.p_ctrl, g_jpeg_decode0.p_cfg); Initial configuration.
outputBufferSet	g_jpeg_decode0.p_api->outputBufferSet(g_jpeg_decode0.p_ctrl, p_buffer, buffer_size); Assign output buffer to JPEG codec for storing output data.
horizontalStrideSet	g_jpeg_decode0.p_api->horizontalStrideSet(g_jpeg_decode0.p_ctrl, stride); Configure the horizontal stride value.
imageSubsampleSet	g_jpeg_decode0.p_api->imageSubsampleSet(g_jpeg_decode0.p_ctrl, horizontal, vertical); Configure the horizontal and vertical subsample settings.
inputBufferSet	g_jpeg_decode0.p_api->inputBufferSet(g_jpeg_decode0.p_ctrl, p_buffer, size); Assign input data buffer to JPEG codec.

linesDecodedGet	g_jpeg_decode0.p_api->linesDecodedGet g_jpeg_decode0.p_ctrl, p_lines); Return the number of lines decoded into the output buffer.
imageSizeGet	g_jpeg_decode0.p_api->imageSizeGet(g_jpeg_decode0.p_ctrl, p_horizontal, p_vertical); Retrieve image size during decoding operation.
statusGet	g_jpeg_decode0.p_api->statusGet(g_jpeg_decode0.p_ctrl, p_status); Retrieve current status of the JPEG codec module.
close	g_jpeg_decode0.p_api->close(g_jpeg_decode0.p_ctrl); Cancel an outstanding operation.
versionGet	g_jpeg_decode0.p_api->versionGet(&version); Get version and store it in provided pointer p_version.
pixelFormatGet	g_jpeg_decode0.p_api->pixelFormatGet(g_jpeg_decode0.p_ctrl, p_color_space); Get the input pixel format.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INVALID_ALIGNMENT	Horizantal stride is not 8-byte aligned.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to change during codec operation. Or set output buffer first.
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG codec.
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	Invalid buffer size.
SSP_ERR_INVALID_MODE	JPEG Codec module is not decoding.

SSP_ERR_IMAGE_SIZE_UNKNOWN	The image size is unknown. More input data may be needed.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.30.3 JPEG Decode HAL Module Operational Overview

The JPEG Decoder HAL module can be used in the Input Buffer Streaming mode or JPEG Output Buffer Streaming mode.

Input Buffer Streaming Mode Operational Description

In this scenario the JPEG image data resides on a file, or is received from network. The HAL-layer driver is able to handle this scenario without requiring the input data to be stored in memory first.

Output Buffer Streaming Mode Operational Description

In this scenario the application needs to write the decoded image data to a file or to a network. The HAL-layer driver does not require the application to allocate memory for the entire frame. Instead the application may choose to decode one or more lines at a time. With this feature the amount of memory needed for the output data is greatly reduced.

MJPEG Decode Operational Description

In this scenario the application needs to display a continuous stream of JPEG images on the native display. The JPEG images can reside on a file or can be received from the network. The HAL driver handles this scenario by using the Input Buffer Streaming Mode feature of the JPEG Decode module.

The basic flow to achieve this would be:

1. Open the JPEG Decode driver.
2. Set the image parameters: horizontal stride, image sub-samples.
3. Set the input buffer which holds the jpeg image frame.
4. Set the output buffer to hold the raw image.
5. Display the decoded image.
6. Close the jpeg driver.
7. Repeat the process from step 1 as needed.

JPEG Decode HAL Module Important Operational Notes and Limitations

JPEG Decode HAL Module Operational Notes

JPEG Decode Callbacks A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the decoding procedure.

Event Name	Event Condition
JPEG_DECODE_STATUS_ERROR	JPEG Decode module encountered an error.

JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG Decode obtained the image size of data to be decoded, and paused.
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG Decode paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG Decode paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_DONE	JPEG Decode operation has successfully completed.

Note

Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

JPEG Decode HAL Module Limitations

- The JPEG Decode HAL module only support JPEG decoding processing. For encoding, use the JPEG Encode Driver.
- If both drivers are in use, the JPEG Encode driver needs to be closed in order to use the JPEG Decode Driver (or vice versa, as both drivers share the same IP).
- Ensure timeout detection logic is implemented (as in sf_jpeg_decode_api_t::wait API of SF_JPEG_DECODE). If timeout error occurs while decoding the image, close the driver, re-open it and then perform the decoding operation.
- Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.2.30.4 Including the JPEG Decode HAL Module in an Application

This section describes how to include the JPEG Decode HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Decode Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the JPEG Decode Driver is g_jpeg_decode0. This name can be changed in the associated Properties window.)

JPEG Decode HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_jpeg_decode0 JPEG Decode Driver on r_jpeg_decode	Threads	**New Stack> Driver> Graphics> JPEG Decode Driver**

When the JPEG Decode Driver on r_jpeg_decode is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

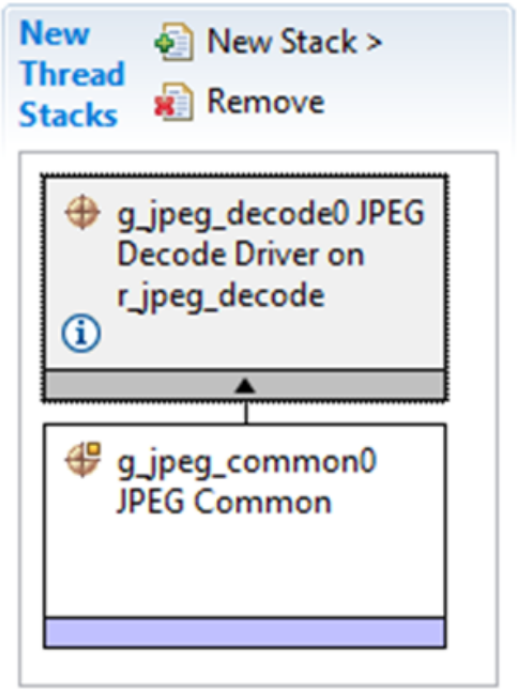


Figure 335: JPEG Decode HAL Module Stack

4.2.30.5 Configuring the JPEG Decode HAL Module

The JPEG Decode HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the JPEG Decode HAL Module on r_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.

Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Decompression interrupt priority selection.

Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Data transfer interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Common Module

ISDE Property	Value	Description
Name	g_jpeg_common0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

JPEG Decode HAL Module Clock Configuration

The JPEG Decode HAL Module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

JPEG Decode HAL Module Pin Configuration

The JPEG Decode HAL Module has no configurable input or output pins.

4.2.30.6 Using the JPEG Decode HAL Module in an Application

The typical steps in using the JPEG Decode HAL module in an application are:

1. Initialize the JPEG Decode using the jpeg_decode_api_t::open API.
2. Set the horizontal stride using the jpeg_decode_api_t::horizontalStrideSet API.
3. Set vertical and horizontal image sub-sample using the jpeg_decode_api_t::imageSubsampleSet API.
4. Set the input buffer address (which contains the JPEG image) using the jpeg_decode_api_t::inputBufferSet API.
5. Set the output buffer (should be large enough to hold the raw image data) using the jpeg_decode_api_t::outputBufferSet API.
6. The jpeg_decode_api_t::statusGet API can be used to get the JPEG operation. It returns an enumerated value (described above) to notify the user.
    a. The status JPEG_DECODE_STATUS_DONE returned from the jpeg_decode_api_t::statusGet API shows that the Decode operation is complete.
    b. The status JPEG_DECODE_STATUS_INPUT_PAUSE or JPEG_DECODE_STATUS_OUTPUT_PAUSE returned from the jpeg_decode_api_t::statusGet API shows that the decode operation is not complete.
    c. Implement a timeout detect function as part of this step. Refer to the wait function used for the JPEG Decode Framework if an illustration is required.
7. Operate on the received raw image data as needed by the application.
8. Close the module using the jpeg_decode_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

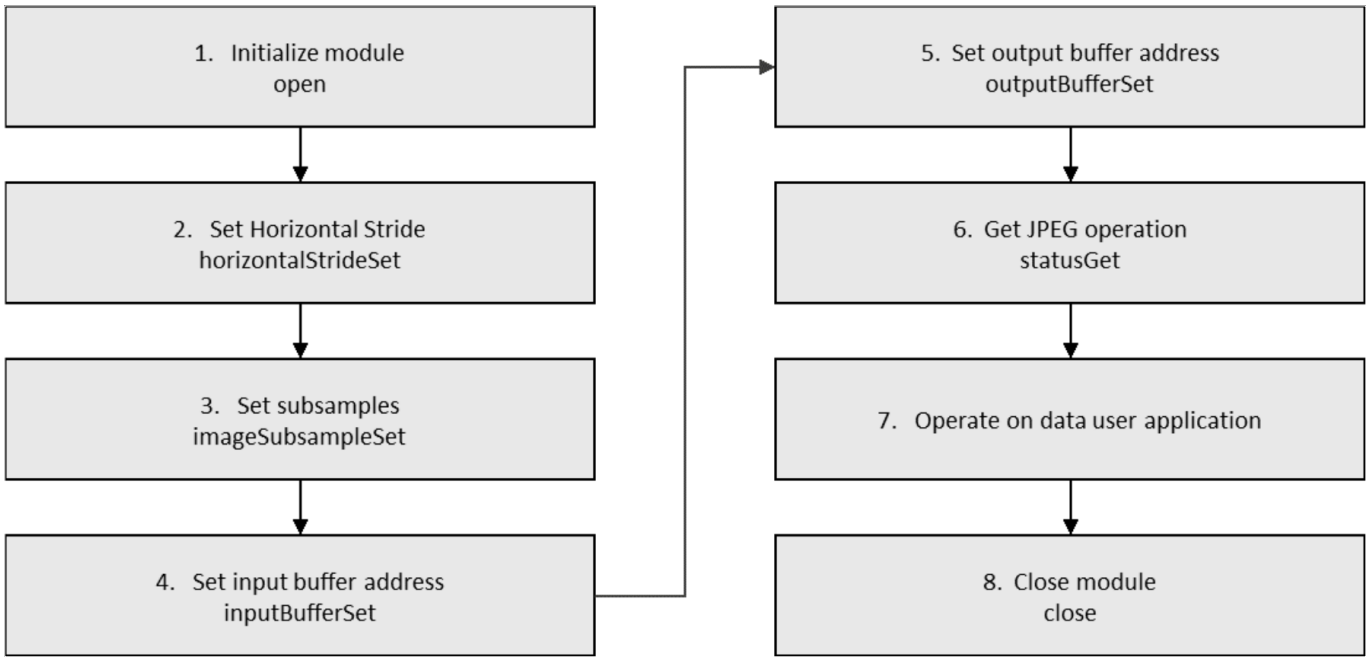


Figure 336: Flow Diagram of a Typical JPEG Decode HAL Module Application

4.2.31 JPEG Encode Driver

4.2.31.1 JPEG Encode HAL Module Introduction

The JPEG Encode HAL module provides a high-level API for industry standard JPEG image encode processing (compression) and uses the Synergy MCU JPEG Codec peripheral. A user callback function is available to inform the application program of key processing events.

JPEG Encode HAL Module Features

- Supports JPEG Compression.
- Supports polling mode that allows an application to wait for JPEG Encoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical resolution, horizontal stride, and Quality factor.
- Supports putting raw image data in an input buffer and an output buffer to store the encoded/compressed jpeg image.
- Supports streaming raw image data into JPEG Encoder module. This feature allows an application to read coded raw image from a capture device or camera or from network without buffering the entire image.
- Only supports the YCbCr422 color space to input.
- Supports DRI Maker for RTP streaming application.
- Supports quality factor configuration: The quality factor value determines the quality of output image.

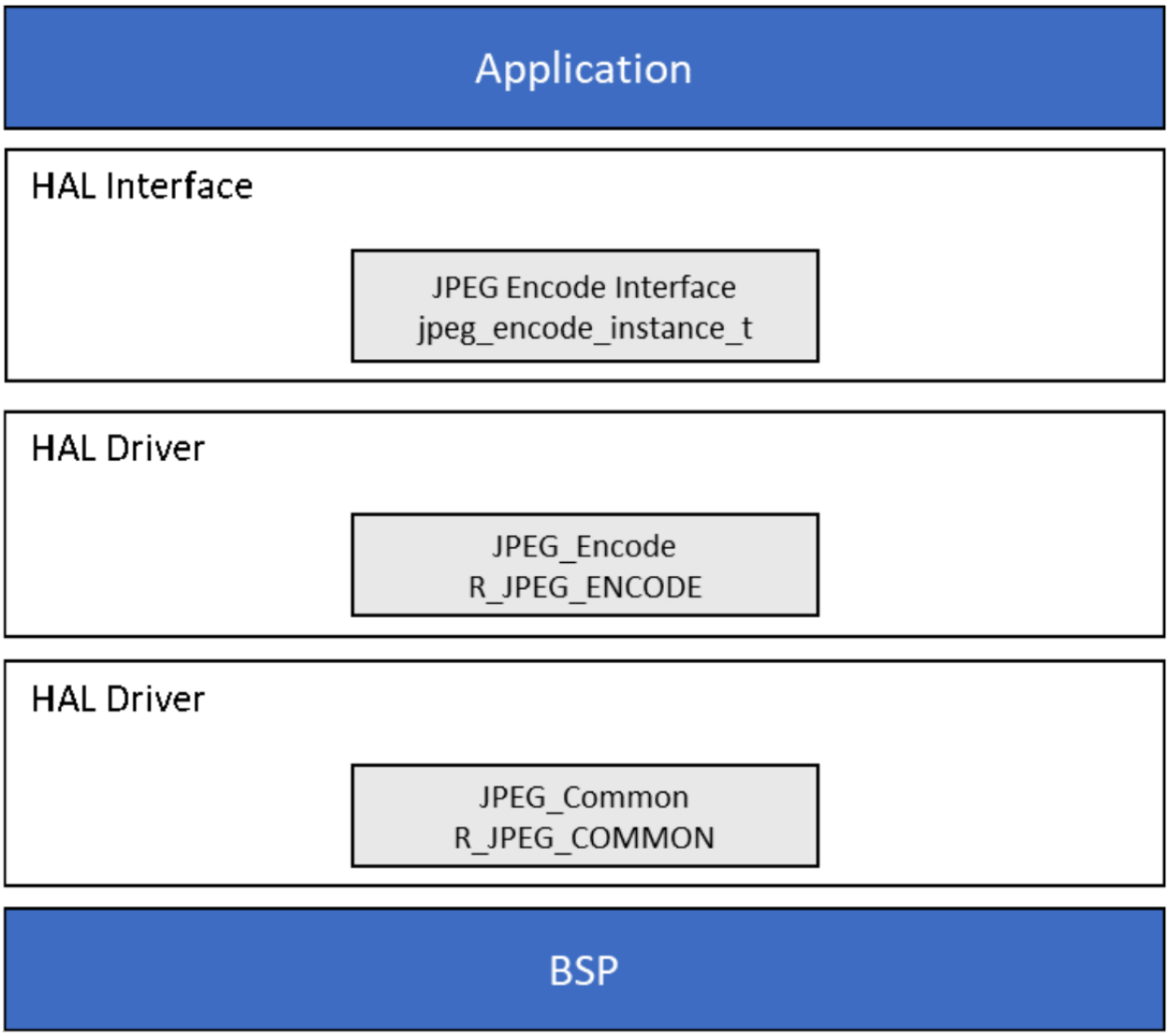


Figure 337: JPEG Encode HAL Module Block Diagram

JPEG Encode Hardware support details

The following hardware features are, or are not, supported by SSP for JPEG.

Legend:

Symbol			Meaning		
✓			Available (Tested)		
☒			Not Available (Not tested/not functional or both)		
N/A			Not supported by MCU		
MCU Group	Input data format 8 lines by 8 pixels in YCbCr444	Input data format 8 lines by 16 pixels in YCbCr422	Input data format 8 lines by 32 pixels in YCbCr411	Input data format 16 lines by 16 pixels in YCbCr420	Output format JPEG

S124	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A
S5D9	N/A	✓	N/A	N/A	✓
S7G2	N/A	✓	N/A	N/A	✓

4.2.31.2 JPEG Encode HAL Module APIs Overview

The JPEG Encode HAL Module defines APIs to open, set up processing parameters for, process, get status from and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the HAL Module API Summary.

JPEG Encode HAL Module API Summary

Function Name	Example API Call and Description
open	g_jpeg_encode0.p_api->open(g_jpeg_encode0.p_ctrl, g_jpeg_encode0.p_cfg); Initial configuration.
imageParameterSet	g_jpeg_encode0.p_api->imageParameterSet(g_jpeg_encode0.p_ctrl, p_image_parameters); Set image parameters to JPEG Codec.
outputBufferSet	g_jpeg_encode0.p_api->outputBufferSet(g_jpeg_encode0.p_ctrl, p_buffer); Assign output buffer to JPEG Codec for storing output data.
inputBufferSet	g_jpeg_encode0.p_api->inputBufferSet(g_jpeg_encode0.p_ctrl, p_buffer, buffer_size); Assign input data buffer to JPEG Codec.
statusGet	g_jpeg_encode0.p_api->statusGet(g_jpeg_encode0.p_ctrl, p_status); Retrieve current status of the JPEG Codec module.
close	g_jpeg_encode0.p_api->close(g_jpeg_encode0.p_ctrl); Cancel an outstanding operation.

versionGet	g_jpeg_encode0.p_api->versionGet(&version); Get version and store it in the provided pointer version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to change during codec operation. Or set output buffer first.
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG codec.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.31.3 JPEG Encode HAL Module Operational Overview

The JPEG Encoder HAL module can be used in the input buffer streaming mode or normal mode.

Input Buffer Streaming Mode Operational Description

In this mode the raw image data arrives from the network, file or capturing device in separate data 'chunks'. The HAL-layer driver handles this mode without requiring the input data to be stored in memory first.

Normal Operational Description

In this mode raw image data arrives from the network, file or capturing device as a complete frame. The HAL-layer driver handles this mode and can compress the entire raw image frame.

JPEG Encode HAL Module Important Operational Notes and Limitations

JPEG Encode HAL Module Operational Notes

Motion JPEG

There is no defined standard for Motion JPEG (MJPEG), but the basic concept is to continuously project JPEG images to the rendering device. The basic application steps to implement an MJPEG function using the JPEG Encoder HAL module are as follows:

1. Open the JPEG Encoder driver with the desired quality factor value (default = 50).
2. Setup the image resolution (optional if already configured in Thread stack window).
3. Initialize the capturing device for capturing a YCbCr422 image.
4. Set the output buffer to hold the jpeg image using the jpeg_encode_api_t::outputBufferSet API function.
5. Capture the image.
6. Set the input buffer which holds the RAW YCbCr 422 image captured from the capturing device using jpeg_encode_api_t::inputBufferSet API function.
7. Check the status of the encode operation and if DONE send the image to the rendering device.
8. Continue the process from step 5 as needed.

JPEG Encode Callbacks

A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the encoding procedure.

Event Name	Event Condition
JPEG_ENCODE_STATUS_INPUT_PAUSE	JPEG Encode paused waiting for more input data.
JPEG_ENCODE_STATUS_DONE	JPEG Encode operation has successfully completed.

Note

Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

JPEG Encode HAL Module Limitations

- The JPEG Encode HAL module only support JPEG Encode processing. For decoding please use the JPEG Decode HAL module.
- In an Application where both the encode and decode modules are used, JPEG Decode module needs to be closed for the application to use JPEG Encode driver (or vice versa) as both modules share the same MCU peripheral.
- The JPEG Encode HAL module only supports the "Normal byte order" in the thread stack window. Other option may result in an invalid image.

4.2.31.4 Including the JPEG Encode HAL Module in an Application

This section describes how to include the JPEG Encode HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Encode Driver to an application, simply add it to a thread using the stacks selection

sequence given in the following table. (The default name for the JPEG Encode Driver is g_jpeg_encode0. This name can be changed in the associated Properties window.)

JPEG Encode HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_jpeg_encode0 JPEG Encode Driver on r_jpeg_encode	Threads	**New Stack> Driver> Graphics> JPEG Encode Driver**

When the JPEG Encode Driver on r_jpeg_encode is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

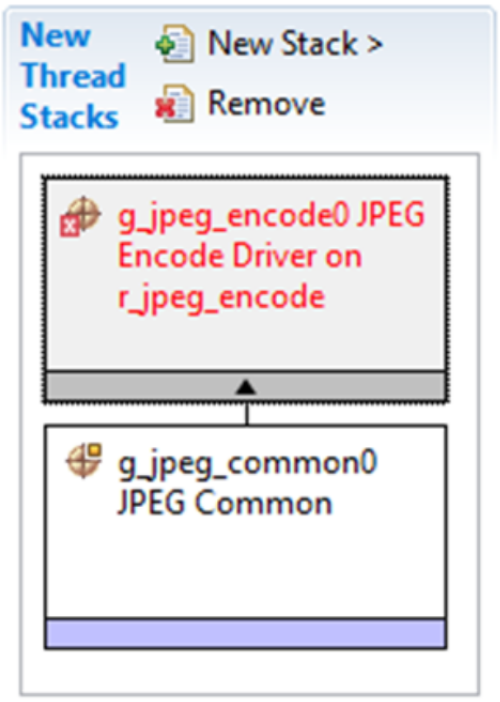


Figure 338: JPEG Encode HAL Module Stack

4.2.31.5 Configuring the JPEG Encode HAL Module

The JPEG Encode HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and

makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the JPEG Encode HAL Module on r_jpeg_encode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_encode0	Module name.
RAW Image Vertical Resolution	800	RAW image vertical resolution selection.
RAW Image Horizontal Resolution	480	RAW image horizontal resolution selection.
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Byte order for input data format selection.

Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Byte order for output data format selection.
Define Restart Marker	512	Define restart marker selection.
Quality Factor	50	Quality factor selection.
Name of user callback function	NULL	Name of user callback function selection.
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Decompression interrupt priority selection.
Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Data transfer interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Common Module

ISDE Property	Value	Description
Name	g_jpeg_common0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

JPEG Encode HAL Module Clock Configuration

The JPEG Encode HAL Module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

JPEG Encode HAL Module Pin Configuration

The JPEG Encode HAL module has no configurable input or output pins on the device.

4.2.31.6 Using the JPEG Encode HAL Module in an Application

The typical steps in using the JPEG Encode HAL module in an application are:

1. Initialize the JPEG Encode using jpeg_encode_api_t::open API.
    a. Configures the quality factor, horizontal stride, horizontal and vertical resolution.
2. Set the output buffer address (should be large enough to hold compressed jpeg image) using the jpeg_encode_api_t::outputBufferSet API.
3. Set the input buffer to start the encoding operation (address of raw image data and size) using thejpeg_encode_api_t::inputBufferSet API.
4. The jpeg_encode_api_t::statusGet API can be used to get the JPEG operation, thejpeg_encode_api_t::statusGet API return an enumerated value (described above) to notify user.
    a. Status JPEG_ENCODE_STATUS_DONE from the jpeg_encode_api_t::statusGet API shows that the encode operation is complete.
    b. Status JPEG_ENCODE_STATUS_INPUT_PAUSE from the jpeg_encode_api_t::statusGet API shows that driver is waiting for more input – Go to step 3 and set the input buffer with remaining data.
5. Operate on the received JPEG image data as needed by the application.
6. Close the module using the jpeg_encode_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

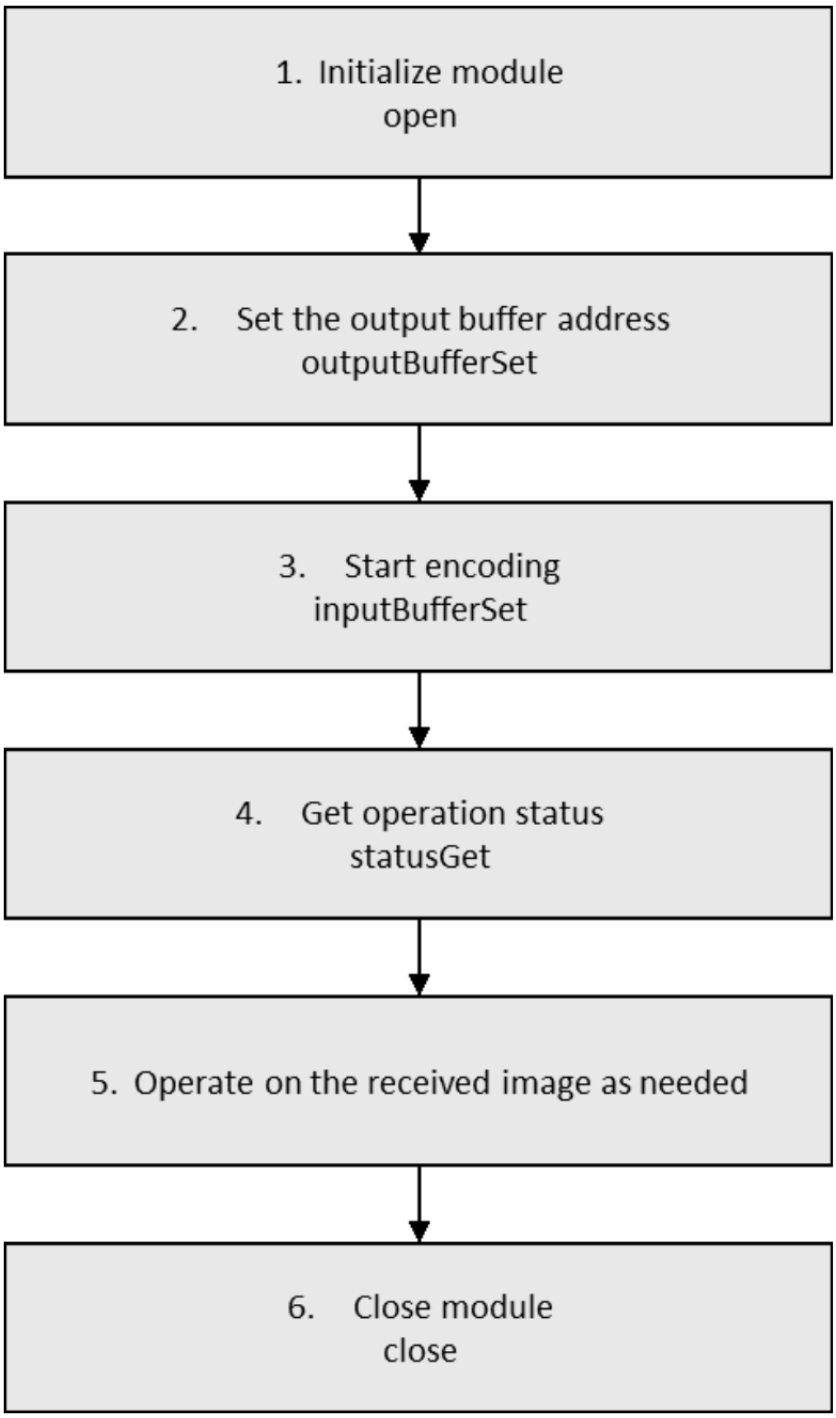


Figure 339: Flow Diagram of a Typical JPEG Encode HAL Module Application

4.2.32 Key Matrix Driver

4.2.32.1 Key Matrix HAL Module Introduction

The Key Matrix HAL module provides a high-level API for key input applications and uses the key-interrupt function peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU of a key press event.

Key Matrix HAL Module Features

This Key Matrix HAL module configures and controls the Key Interrupt (KINT) peripheral. It implements the following key functions:

- Supports both rising and falling edges on KINT channels
- Supports interrupt-based event notification
- Supports a bit-masking function to capture multiple events efficiently
- Supports a matrix keypad with edges on any two channels

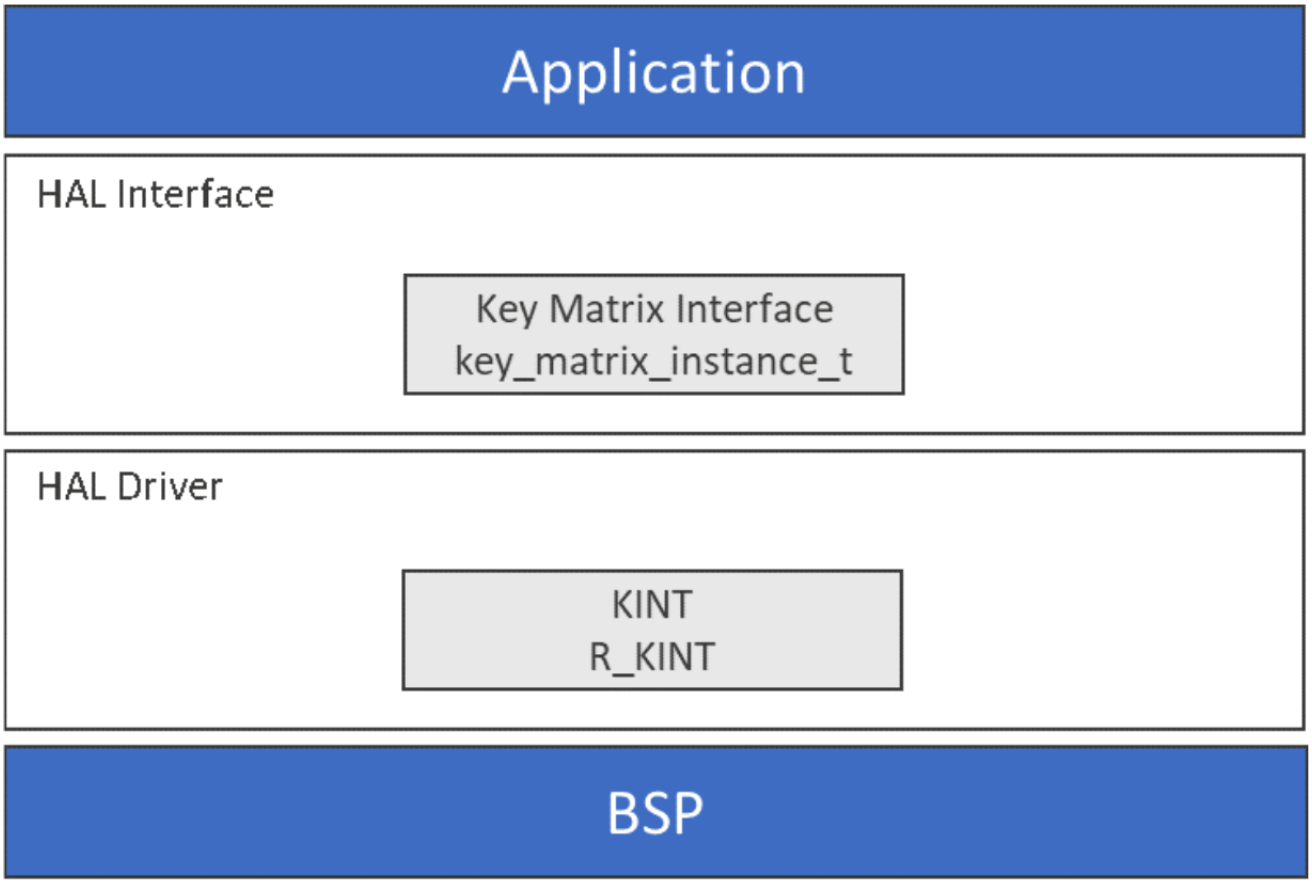


Figure 340: Key Matrix HAL Module Block Diagram

KINT Hardware support details

The following hardware features are, or are not, supported by SSP for KINT.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A	Not supported by MCU

MCU Group	Vary Input from KR00 to KR07
S124	✓
S128	✓
S1JA	✓
S3A1	✓
S3A3	✓
S3A6	✓
S3A7	✓
S5D3	✓
S5D5	✓
S5D9	✓
S7G2	✓

4.2.32.2 Key Matrix HAL Module APIs Overview

The Key Matrix HAL module defines APIs for opening, closing, enabling and disabling key-interrupt functions. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Key Matrix HAL Module API Summary

Function Name	Example API Call and Description
open	g_keymatrix_on_kint.p_api->open(g_kint.p_ctrl, g_kint.p_cfg_cfg) Initial configuration.
enable	g_keymatrix_on_kint.p_api->enable(g_kint.p_ctrl) Enable Key interrupt.
disable	g_keymatrix_on_kint.p_api->disable(g_kint.p_ctrl) Disable Key interrupt.
triggerSet	g_keymatrix_on_kint.p_api->triggerSet() Set trigger for Key interrupt.
close	g_keymatrix_on_kint.p_api->close(&g_keymatrix) Allow driver to be reconfigured. May reduce power consumption.

versionGet	g_keymatrix_on_kint.p_api->versionGet(&version) Get version and store it in provided pointer version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully completed.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_ARGUMENT	Argument is invalid.
SSP_ERR_HW_LOCKED	The API has already been opened. It must be closed before it can be opened again.
SSP_ERR_NOT_OPEN	The peripheral is not opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.32.3 Key Matrix HAL Module Operational Overview

The Key Matrix HAL module configures the Key Interrupt (KINT) peripheral to detect rising or falling edges on any of the KINT channels. When such an event is detected on any of the configured pins, the module generates an interrupt; the interrupt then calls the user callback (p_callback) with the callback argument keymatrix_callback_args_t that specifies the channel(s) on which the edge was detected using a bitmask.

Even though detection of an edge on any one channel generates the interrupt, the callback returns a bitmask keymatrix_channels_t of all the pins that were triggered at that time (if any other pins also detected an edge). Thus, an interrupt is not necessarily generated for edge detection on each pin if an edge was also detected on another pin before the callback was called. If a new edge is detected after the callback was called, then the interrupt is triggered again, resulting in a new callback. The bit mask in the user callback should be checked to identify the interrupt source channels.

This module can be used to implement a matrix keypad with edges on any two channels indicating the actual key that was pressed; alternatively, the module can be used as a single input to detect an edge on an input pin.

Key Matrix HAL Module Important Operational Notes and Limitations

Key Matrix HAL Module Operational Notes

- To trigger a transfer of data using the DMAC or DTC peripheral when a trigger edge is detected, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_KEY_INT.
- The KINT module can trigger the start of other peripherals available to the ELC. For details,

see the ELC User's Guide in the SSP User's Manual.
- You must enable the KINT (INTKR) interrupt in the BSP for this module to operate, regardless of whether a callback is used in the keymatrix_api_t::open call.

Key Matrix HAL Module Limitations

- This module does not support polling-mode operation.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.32.4 Including the Key Matrix HAL Module in an Application

This section describes how to include the Key Matrix HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Key Matrix Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Key Matrix Driver is g_kint0. This name can be changed in the associated Properties window.)

Key Matrix HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_kint0 Key Matrix Driver on r_kint	Threads	**New Stack**> **Driver**> **Input**> **Key Matrix Driver on r_kint**

When the Key Matrix Driver on r_kint is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

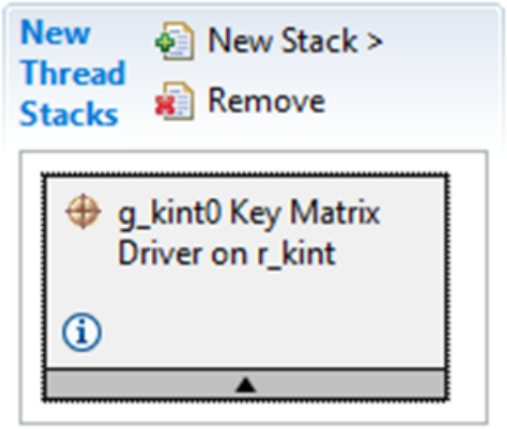


Figure 341: Key Matrix HAL Module Stack

4.2.32.5 Configuring the Key Matrix HAL Module

The Key Matrix HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Key Matrix HAL Module on r_kint

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_kint0	Module name.
Channel 0-7	Unused, Used Default: Unused	This is a bit-mask with each bit specifying if that channel is to be enabled or not. Select the channels to be used.

Trigger Type	Rising Edge, Falling Edge Default: Rising Edge	Specify if the enabled channels detect a rising edge or a falling edge. Note: Either all channels detecting a rising edge or all channels detecting a falling edge.
Interrupt enabled after initialization	True, False Default: False	Specify if the module interrupts must be enabled as part of the open call.
Callback	NULL	A user callback function can be registered in keymatrix_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time a configured edge is detected on any of the channels. Note: Without callback, the application cannot determine whether an event has occurred. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Key Matrix HAL Module Clock Configuration

The Key Matrix HAL module does not require a specific clock configuration.

Key Matrix HAL Module Pin Configuration

The KINT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the KINT pins:

Pin Selection for the Key Matrix HAL Module on r_kint

Resource	ISDE Tab	Pin selection Sequence
KINT	Pins	**Select Peripherals> Input:KINT> KINT0.**

Note

The selection sequence assumes KINT0 is the desired hardware target for the driver.

Pin Configuration Settings for the Key Matrix HAL Module on r_kint

Property	Value	Description
Operation Mode	Disabled, Custom Default: Disabled	Select Custom as the Operation Mode.
KRM0:7	None, Pnn Default: None	Key Interrupt Pin selection.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.32.6 Using the Key Matrix HAL Module in an Application

The typical steps in using the Key Matrix HAL module in an application are:

1. Initialize the Key Matrix HAL module using the keymatrix_api_t::open API.
2. If the autostart configuration setting is true, the module starts operation immediately.
    a. If the autostart is not set, use keymatrix_api_t::enable API to enable operation.
3. Respond to key inputs.
4. Disable operation using the keymatrix_api_t::disable API.
5. To modify trigger edge after initialization, use the keymatrix_api_t::triggerSet API.
6. Close the module by using the keymatrix_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

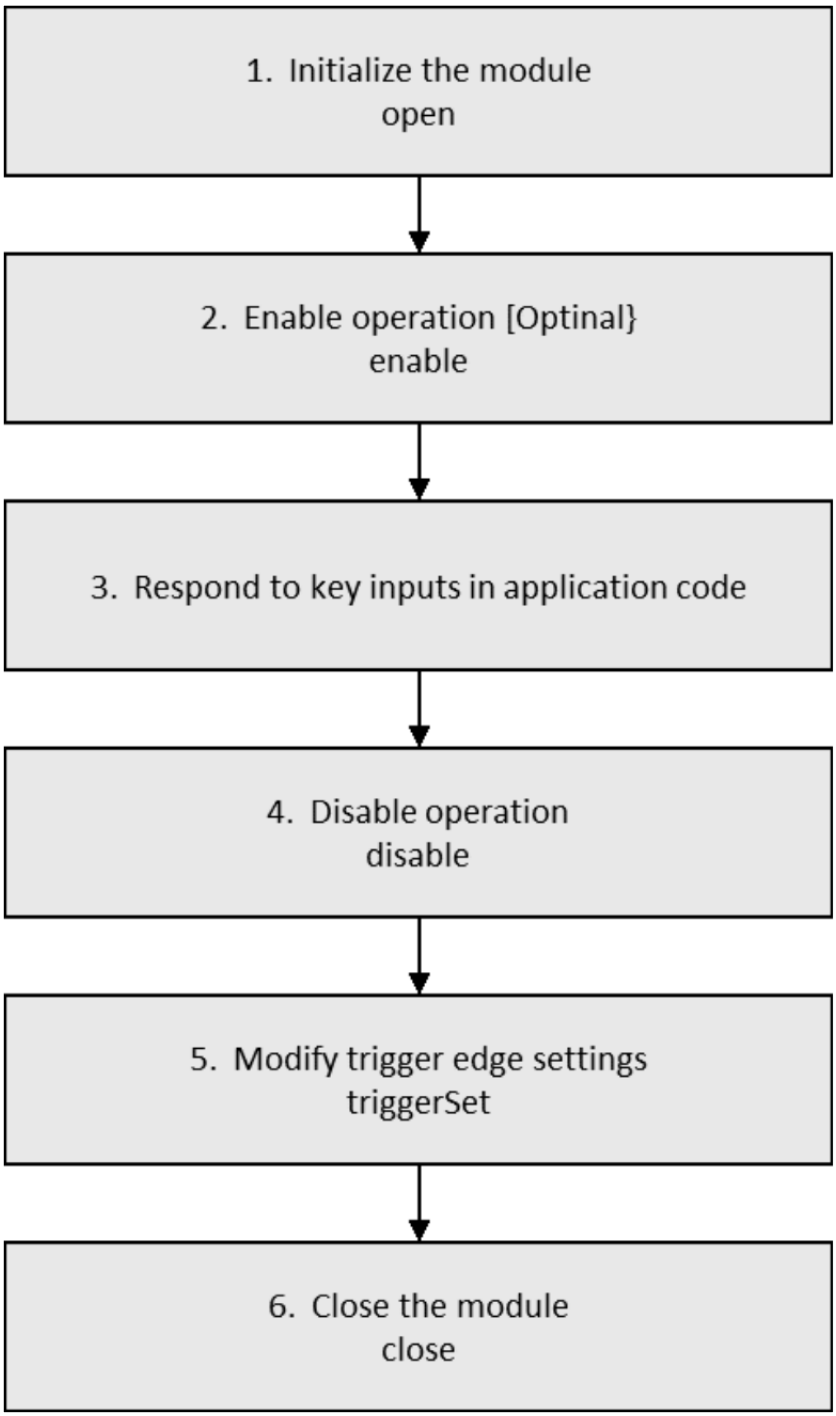


Figure 342: Flow Diagram of a Typical Key Matrix HAL Module Application

4.2.33 Low Power Modes Driver on r_lpmv2

4.2.33.1 LPM V2 HAL Module Introduction

The Low Power Modes V2 HAL module provides a high-level API for low-power mode applications and uses the low-power mode hardware peripheral on the Synergy MCU.

LPM V2 HAL Module Features

- Supports configuration of MCU operating power-control modes and MCU low-power modes
- Supports the following low power modes:
  - Deep Software Standby mode
  - Software Standby mode
  - Sleep mode
  - Snooze mode
- Supports reducing power consumption when in deep stand-by mode through internal power-supply control and by resetting the states of I/O ports.
- Supports disabling and enabling of the MCU's other hardware peripherals.

Note

Not all low-power V2 modes are available on all MCU Groups.

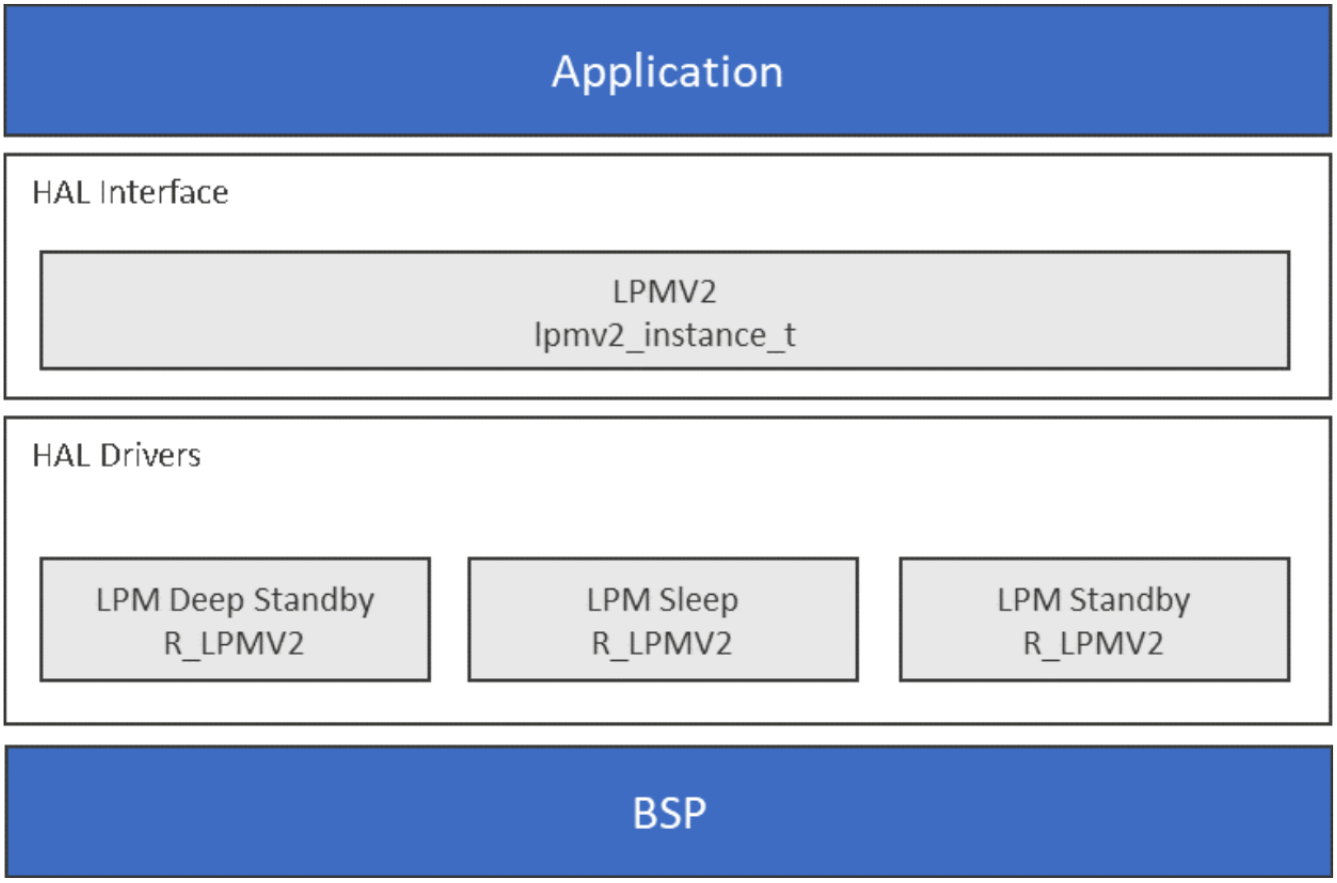


Figure 343: LPM V2 HAL Module Block Diagram

LPMV2 Hardware support details

The following hardware features are, or are not, supported by SSP for LPM V2:

Legend:

Symbol	Meaning

✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Sleep Low Power Mode	Software Standby Low Power Mode	Deep Standby Low Power Mode	Snooze enabled in Software Standby Low Power Mode	Snooze Linking using ELC	DTC state in Snooze Mode
S124	✓	✓	N/A	✓	☒	✓
S128	✓	✓	N/A	✓	☒	✓
S1JA	✓	✓	N/A	✓	☒	✓
S3A1	✓	✓	N/A	✓	☒	✓
S3A3	✓	✓	N/A	✓	☒	✓
S3A6	✓	✓	N/A	✓	☒	✓
S3A7	✓	✓	N/A	✓	☒	✓
S5D3	✓	✓	✓	✓	☒	✓
S5D5	✓	✓	✓	✓	☒	✓
S5D9	✓	✓	✓	✓	☒	✓
S7G2	✓	✓	✓	✓	☒	✓

MCU Group	State of address bus and bus signals in Standby or Deep Standby Mode	Enter Snooze mode via RXD0 (SCI0)	IO Port state control after waking up from Deep Standby Mode	Internal Power Supply control in Deep Standby Mode (power supply to LOCO, Standby SRAM, AGTn, and USBHS/FS)
S124	☒	✓	N/A	☒
S128	☒	✓	N/A	☒
S1JA	☒	✓	N/A	☒
S3A1	✓	✓	N/A	☒
S3A3	✓	✓	N/A	☒
S3A6	✓	✓	N/A	☒
S3A7	✓	✓	N/A	☒
S5D3	✓	✓	✓	✓
S5D5	✓	✓	✓	✓
S5D9	✓	✓	✓	✓

S7G2	✓	✓	✓	✓

4.2.33.2 LPM V2 HAL Module APIs Overview

The Low Power Modes V2 HAL module defines APIs for configuring operations and enabling and disabling low-power operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Note

The Low Power Modes V2 HAL module will no longer handle operating power-control modes of the MCU; these are now handled by the CGC HAL module.

The following API examples illustrate sleep-mode use; "deep_standby" and "standby" can be substituted for "sleep" in the API examples to create examples for those modes.

LPM V2 HAL Module API Summary

Function Name	Example API Call and Description
init	g_lpmv2_sleep0.p_api->init(g_lpmv2_sleep0.p_cfg); Open the LPM driver module Initialized the LPM block according to the passed in config structure.
lowPowerCfg	g_lpmv2_sleep0.p_api->lowPowerCfg(power_mode, output_port_enable, power_supply, io_port_state); Configure a low power mode.
lowPowerModeEnter	g_lpmv2_sleep0.p_api->lowPowerModeEnter(void); Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.
versionGet	g_lpmv2_sleep0.p_api->versionGet(&version); Get the driver version and place it at the pointer version.
clearIOKeep	g_lpmv2_sleep0.p_api->clearIOKeep(void); Clear the IOKEEP bit after deep software stand by mode exit

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.

SSP_ERR_INVALID_POINTER	Pointer is NULL.
SSP_ERR_INVALID_MODE	Invalid settings for specified mode.
SSP_ERR_INVALID_HW_CONDITION	OPCMTSF and SOPCMTSF flags are not cleared within internally set timeout.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.33.3 LPM V2 HAL Module Operational Overview

LPM V2 Initialization

The LPM V2 API function lpmv2_api_t::init should be called before calling any other LPM V2 function. The init function handles initialization of internal variables and locks.

Sleep Low-Power Mode

By default, at power on, sleep mode is enabled as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. Any interrupt wakes the MCU device from sleep low-power mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the Systick interrupt used by the ThreadX® thread scheduler. The LPM API function lpmv2_api_t::init should be called before any other function. The LPM API function, lpmv2_api_t::lowPowerCfg, can be used to configure the MCU to use sleep as its low-power mode. The LPM API function lpmv2_api_t::lowPowerModeEnter should be used to directly enter sleep mode.

The following code illustrates configuring sleep as a low-power mode and entering the low-power sleep mode. In this illustration, the LPM V2 sleep module uses the name g_lpmv2_sleep0:

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
    ssp_err_t error = SSP_SUCCESS;
    /* Initialize the LPM V2 Driver */
    error = g_lpmv2_sleep0.p_api->init();
    /* Handle error if any */
    /* Configure LPM peripheral for sleep mode */
    error = g_lpmv2_sleep0.p_api->lowPowerCfg(g_lpmv2_sleep0.p_cfg);
    /* Handle error if any */
    /* Entry sleep mode */
```

```
    error = g_lpmv2_sleep0.p_api->lowPowerModeEnter();
    /* Handle error if any */
}
```

Software Standby Mode for LPM V2

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. Retained are the contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports. Software-standby mode allows significant reduction in power consumption, because most of the oscillators are stopped in this mode. Like Sleep mode, Standby mode requires an interrupt or event be configured and enabled to wake from Standby mode.

The possible triggers for waking from standby mode are enumerated in the Properties window for convenience; multiple triggers can be enabled.

The following code illustrates configuring standby as the low-power mode and entering the low-power standby mode. In this illustration, the LPM V2 standby module with nameg_lpmv2_standby0 is used. A version of this illustration using the Standby module with snooze enabled would be identical:

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
    ssp_err_t error = SSP_SUCCESS;
    /* Initialize the LPM V2 Driver */
    error = g_lpmv2_standby0.p_api->init();
    /* Handle error if any */
    /* Configure LPM peripheral for standby mode */
    error = g_lpmv2_standby0.p_api->lowPowerCfg(g_lpmv2_standby0.p_cfg);
    /* Handle error if any */
    /* Entry standby mode */
    error = g_lpmv2_standby0.p_api->lowPowerModeEnter();
    /* Handle error if any */
}
```

Snooze Mode with Software Standby Mode for LPM V2

Snooze mode is available through the standby mode LPM V2 instance. Choose "Standby with Snooze Enabled" for "Choose the low power mode" in the Properties window. Snooze mode can be used with

some MCU peripherals to execute basic tasks, while keeping the MCU in a low-power state. The snooze settings are below the standby settings in the Properties window. The ADC, DTC, and other peripherals can be enabled in snooze mode. All the settings for snooze are available through configuration properties for the standby instance Snooze is considered an advanced feature.

There are three ways to exit from Snooze mode:

1. Select **Snooze Exit Sources** from the configurator (Transit from Snooze to Software Standby mode).
2. Select **Standby/Snooze Exit Sources** from the configurator (Transit from Snooze to Normal mode).
3. Select **Additional snooze exit sources** from the configurator (Transit from Snooze to Normal mode).

The **Snooze Mode Settings** are only used if the low-power mode choice is **Standby with Snooze Enabled**,"as shown in the following screen capture:

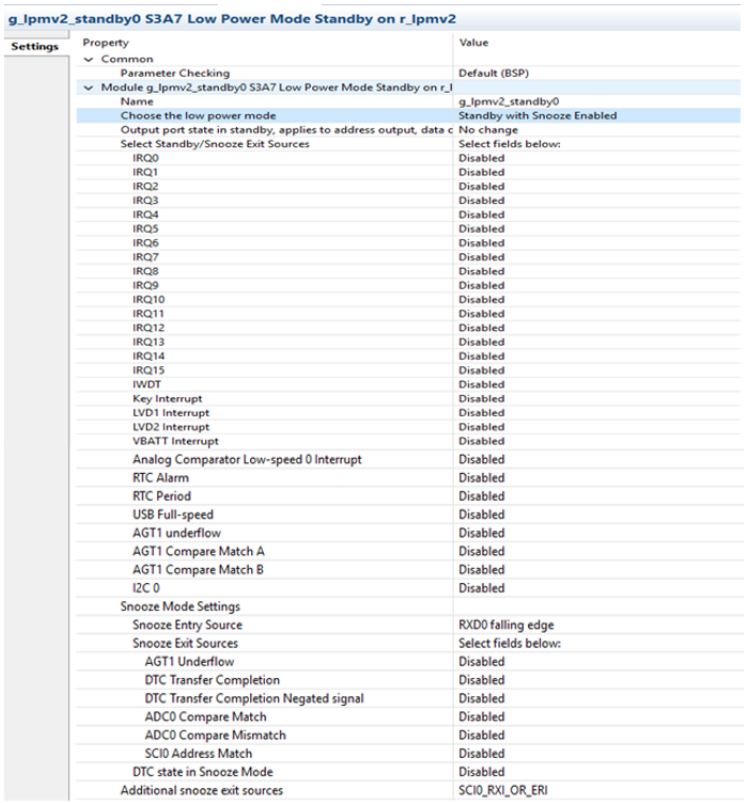


Figure 344: LPM V2 HAL Module Standby Mode With Snooze Setting Enabled

Snooze is a feature of Standby mode that allows some peripherals to run even though the MCU core is not executing instructions. The low-power mode peripheral options related to Snooze mode are shown in the following image. Only one snooze-entry source can be enabled; multiple snooze-exit sources can be enabled. The DTC peripheral can be enabled in snooze mode as well.

Snooze Mode Settings	
Snooze Entry Source	RXD0 falling edge
Snooze Exit Sources	Select fields below:
AGT1 Underflow	Disabled
DTC Transfer Completion	Disabled
DTC Transfer Completion Negated signal	Disabled
ADC0 Compare Match	Disabled
ADC0 Compare Mismatch	Disabled
SCI0 Address Match	Disabled
DTC state in Snooze Mode	Disabled

Figure 345: LPM V2 HAL Module Snooze Mode Settings

Deep Software Standby Mode for LPM V2

Deep Software Standby Mode is only available on some MCU devices. The MCU device always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of a set of wake up events displayed in the configuration Properties window for the LPM deep standby instance.

The possible triggers for waking from deep standby mode are enumerated in the Properties window for convenience. Multiple triggers can be enabled. Some triggers have an associated edge type, falling or rising. These options are enumerated also as shown above and below.

The following illustration is for configuring deep standby as the low power mode and entering low power Deep Standby mode. In this illustration, the LPM V2 Deep Standby module with nameg_lpmv2_deep_standby0 is used:

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
    ssp_err_t error = SSP_SUCCESS;
    /* Initialize the LPM V2 Driver */
    error = g_lpmv2_deep_standby0.p_api->init();
    /* Handle error if any */
    /* Configure LPM peripheral for deep sleep mode */
    error = g_lpmv2_deep_standby0.p_api->lowPowerCfg(g_lpmv2_deep_standby0.p_cfg);
    /* Handle error if any */
    /* Entry deep sleep mode */
    error = g_lpmv2_deep_standby0.p_api->lowPowerModeEnter();
    /* Handle error if any */
```

LPM V2 HAL Module Important Operational Notes and Limitations

LPM V2 HAL Module Operational Notes

Using this driver to configure the LPM peripheral to wake the MCU from standby mode through interrupts requires the interrupt to be configured and enable by the peripheral driver or framework that uses that interrupt. For example, to wake from standby through AGT1 underflow, that interrupt must be enabled through the configuration of the AGT timer module.

If the main oscillator or PLL with main-oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC HAL module properties. See the Wakeup Timing and Duration table in Electrical Characteristics for more information.

When a project uses ThreadX and the low-power mode standby, deep standby, or standby with snooze enabled, the call to the lpmv2_api_t::lowPowerCfg API function should occur immediately before the call to the lpmv2_api_t::lowPowerModeEnter API function. This is necessary since ThreadX also uses low-power modes in its idle loop and tx_thread_sleep function; ThreadX expects the MCU device to be configured for the low-power mode sleep.

When a project uses ThreadX and the low-power mode standby or standby with snooze enabled, the low-power mode should be reverted to sleep after the MCU device wakes from standby after returning from the lpmv2_api_t::lowPowerModeEnter function. This is necessary since ThreadX also uses low-power modes in its idle loop and tx_thread_sleep function; ThreadX expects the MCU device to be configured for the low-power mode sleep. The API function lpmv2_api_t::lowPowerCfg needs to be called again before lpmv2_api_t::lowPowerModeEnter to re-configure the low-power mode to sleep, if tx_thread_sleep() is used in the project, or if there may not always be a thread ready to run.

If the deep software standby mode is used, the configurator provides a property to select whether to reset the IO Ports or maintain the status of IO Ports while coming out of deep software standby. If the property is configured to maintain the status of IO port the status of IO Ports will be maintained, however the application code will have to clear the IOKEEP bit in the DSPBYCR register through an API call to clearIOKeep(), after coming out of the deep software standby mode, to allow the operation on IO Ports thereafter.

Detailed information about the expected power consumption of the MCU device in operating states and in Low Power Modes V2 can be found in the Operating and Standby Current section within the Electrical Characteristics section of the MCU Synergy Hardware User's Manual.

LPM V2 HAL Module Limitations

- Flash stop (code flash disable) is not supported. See the section "Flash Operation Control Register (FLSTOP)" of the S1/S3 Synergy MCU Series Hardware User's Manual.
- Reduced SRAM retention area in software standby mode is not supported. See the section "Power Save Memory Control Register (PSMCR)" of the S3 MCU Series Synergy Hardware User's Manual.
- The MCU may not enter or stay in Software Standby and Deep Software Standby modes with the debugger attached. Instead, the MCU may be woken from Software Standby and Deep Software Standby modes by the debugger. To properly test and verify Software Standby and Deep Software Standby modes, the debugger must not be attached.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC HAL module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.33.4 Including the LPM V2 HAL Module in an Application

This section describes how to include the LPM V2 HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the LPM V2 Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the LPM V2 Driver is g_lpm2_<mode>0. This name can be changed in the associated Properties window.)

LPM V2 HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_lpmv2_deep_standby0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Deep Standby on r_lpmv2**
g_lpmv2_sleep0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Sleep on r_lpmv2**
g_lpmv2_standby0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Standby on r_lpmv2**

When the LPM V2 Driver on r_lpm2 is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

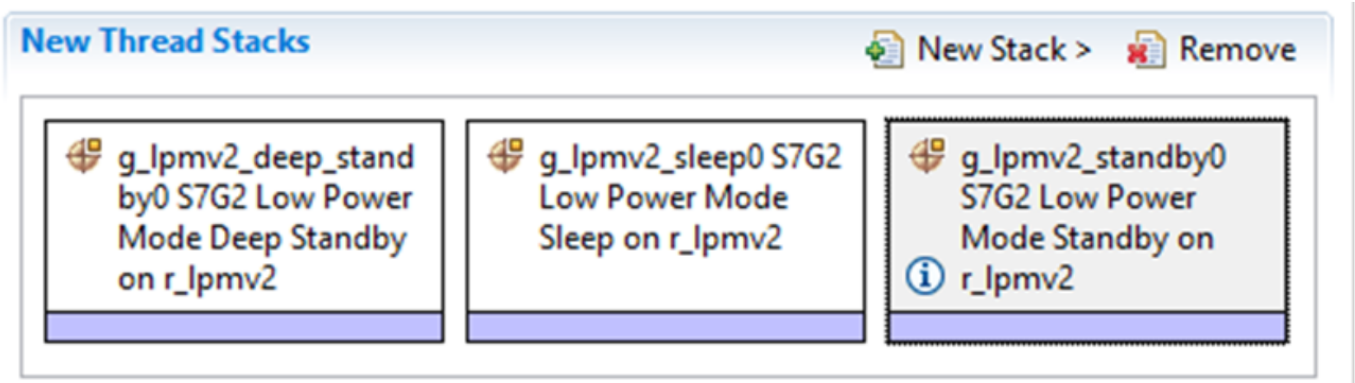


Figure 346: LPM V2 HAL Module Stack

4.2.33.5 Configuring the LPM V2 HAL Module

The LPM V2 HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the LPM Deep Standby Module on r_lpm2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_deep_standby	Module name.
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection.
Maintain or reset the IO port states on exit from deep standby mode	Maintain the IO port states, Reset the IO port states Default: Maintain the IO port states	Maintain/reset I/O port states selection.
Internal power supply control in deep standby mode	Maintain the internal power supply, Cut the power supply to standby RAM, low-speed on-chip oscillator, AGTn, and USPFS/HS resume detecting unit, Cut the power supply to LVDn, standby RAM, low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit Default: Maintain the internal power supply	Internal power supply control selection.
IRQ0-15	Enabled, Disabled Default: Disabled	IRQ0-15 selection.

IRQ0-15 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	IRQ0-15 Edge selection.
LVD1	Enabled, Disabled Default: Disabled	LVD1 selection.
LVD1 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD1 Edge selection.
LVD2	Enabled, Disabled Default: Disabled	LVD2 selection.
LVD2 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD2 Edge selection.
RTC Interval	Enabled, Disabled Default: Disabled	RTC Interval selection.
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection.
NMI	Enabled, Disabled Default: Disabled	NMI selection.
NMI Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	NMI Edge selection.
USBFS	Enabled, Disabled Default: Disabled	USBFS selection.
UBSHS	Enabled, Disabled Default: Disabled	UBSHS selection.
AGT11	Enabled, Disabled Default: Disabled	AGT11 selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the LPM Sleep Module on r_lpm2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking.
Name	g_lpmv2_sleep0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the LPM Standby Module on r_lpm2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_standby0	Module name.
Choose the low power mode	Standby, Standby with snooze Enabled Default: Standby	Low power mode selection.
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection.
Select Standby/Snooze Exit Sources	Select fields below:	Transit from Standby to Normal or Snooze to Normal mode.
IRQ1-15	Enabled, Disabled Default: Disabled	IRQ1-15 selection.
IWDT	Enabled, Disabled Default: Disabled	IWDT selection.
Key Interrupt	Enabled, Disabled Default: Disabled	Key Interrupt selection.
LVD1 Interrupt	Enabled, Disabled Default: Disabled	LVD1 Interrupt selection.
LVD2 Interrupt	Enabled, Disabled Default: Disabled	LVD2 Interrupt selection.
Analog Comparator High-speed 0 Interrupt	Enabled, Disabled Default: Disabled	Analog Comparator High-speed 0 Interrupt selection.

RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection.
RTC Period	Enabled, Disabled Default: Disabled	RTC Period selection.
USB High-speed	Enabled, Disabled Default: Disabled	USB High-speed selection.
USB Full-speed	Enabled, Disabled Default: Disabled	USB Full-speed selection.
AGT1 underflow	Enabled, Disabled Default: Disabled	AGT1 underflow selection.
AGT1 Compare Match A	Enabled, Disabled Default: Disabled	AGT1 Compare Match A selection.
AGT1 Compare Match B	Enabled, Disabled Default: Disabled	AGT1 Compare Match B selection.
12C 0	Enabled, Disabled Default: Disabled	12C 0 selection.
Snooze Entry Source	RXD0 falling edge, IRQ0-IRQ15, KINT, ACMPHS0, RTC Alarm, RTC Period, AGT1 Underflow, AGT1 Compare Match A, AGT1 Compare Match B Default: RXD0 falling edge	Snooze Entry Source selection.
AGT1 Underflow	Enabled, Disabled Default: Disabled	AGT1 Underflow selection.
DTC Transfer Completion	Enabled, Disabled Default: Disabled	DTC Transfer Completion selection.
DTC Transfer Completion Negated Signal	Enabled, Disabled Default: Disabled	DTC Transfer Completion Negated Signal selection.
ADC0 Compare Match	Enabled, Disabled Default: Disabled	ADC0 Compare Match selection.
ADC0 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC0 Compare Mismatch selection.

ADC1 Compare Match	Enabled, Disabled Default: Disabled	ADC1 Compare Match selection.
ADC1 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC1 Compare Mismatch selection.
SCI0 Address Match	Enabled, Disabled Default: Disabled	SCI0 Address Match selection.
DTC state in Snooze Mode	Enabled, Disabled Default: Disabled	DTC state in Snooze Mode selection.
Additional snooze exit sources	SCI0_RXI_OR_ERI Default: SCI0_RXI_OR_ERI	Transit from Snooze to Normal mode.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different states for entering or exiting low-power states.

LPM V2 HAL Module Clock Configuration

The LPM V2 peripheral module does not have any selectable clock sources.

LPM V2 HAL Module Pin Configuration

The LPM V2 peripheral module does not need pin assignments. Pin function selections are done in the properties configuration window.

4.2.33.6 Using the LPM V2 HAL Module in an Application

The typical steps in using the LPM V2 HAL module in an application are:

1. Initialize the Low Power Modes V2 HAL module using the lpmv2_api_t::init API.
2. Configure a low-power mode with the lpmv2_api_t::lowPowerCfg API.
3. Enter a low-power mode with the lpmv2_api_t::lowPowerModeEnter API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

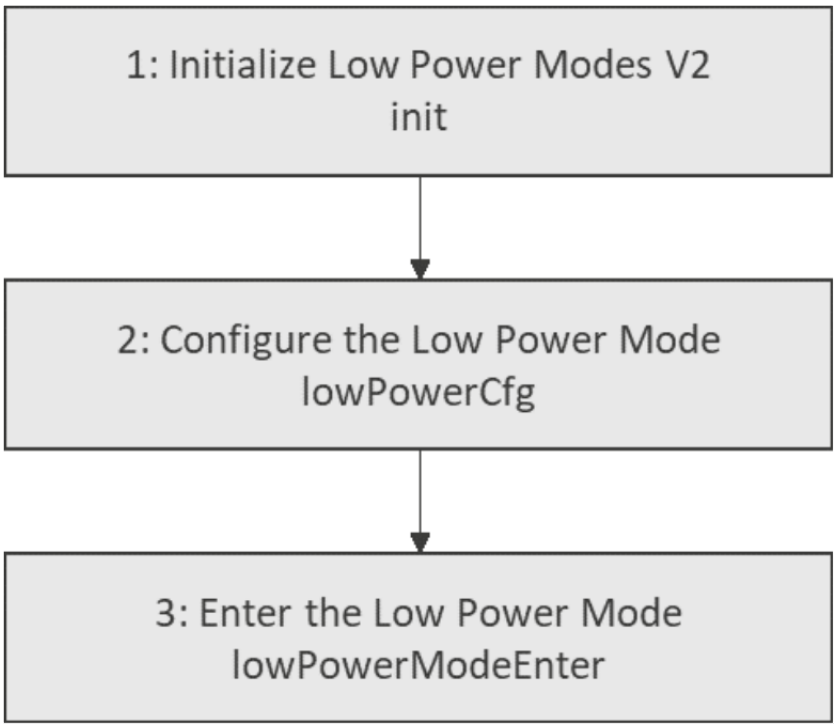


Figure 347: Flow Diagram of a Typical LPM V2 HAL Module Application

4.2.34 Low Voltage Detection Driver

4.2.34.1 LVD HAL Module Introduction

The Low Voltage Detection (LVD) HAL module provides a high-level API for voltage-detection applications and uses the LVD peripheral on the Synergy MCU. A user-defined callback can be created to notify the CPU when a voltage-detection event is triggered. The VCC is the source for all voltage-detection functions.

LVD HAL Module Features

The LVD HAL module supports the following functions:

- V_{CC} as the voltage-detection input
- One build-time configurable low-voltage detector (via OFS1 register)
- Two run-time configurable low-voltage detectors
- Two result flags; one for a threshold check and one for the current state
- Support for both interrupt or polling-event checking

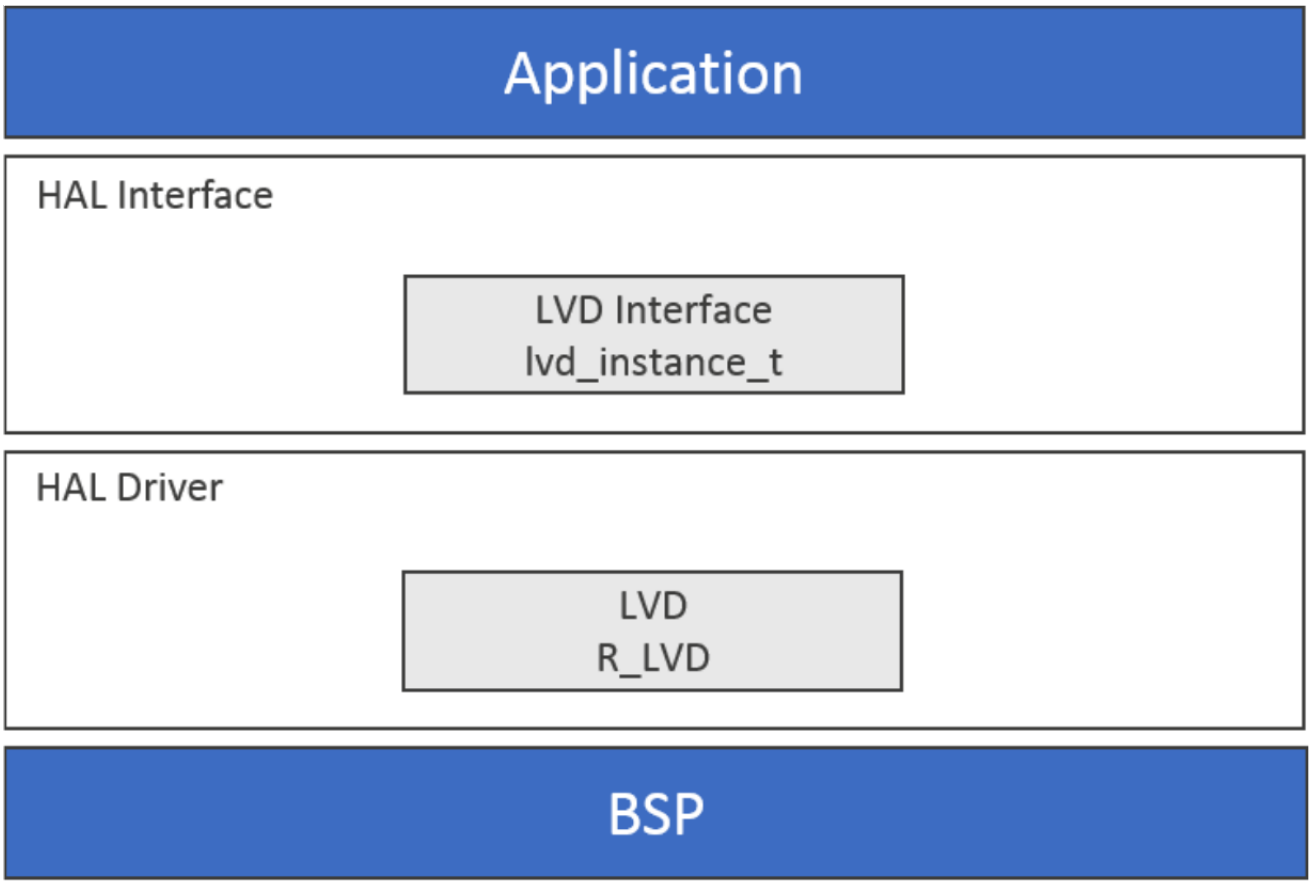


Figure 348: LVD HAL Module Block Diagram

LVD Hardware support details

The following hardware features are, or are not, supported by SSP for LVD:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	VCC Rising voltage generates an interrupt or non maskable interrupt	VCC Falling voltage generates an interrupt or non-maskable interrupt	VCC Rising and falling voltage generates an interrupt or non-maskable interrupt	Callback notification for maskable and non maskable interrupt	Reset on falling voltage	Monitoring LVD 1 and 2 status flags by polling
S124	✓	✓	✓	✓	✓	✓

S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Digital Filtering with adjustable filter time	Event link function through ELC HAL driver
S124	N/A	☒
S128	N/A	☒
S1JA	N/A	☒
S3A1	N/A	☒
S3A3	N/A	☒
S3A6	N/A	☒
S3A7	N/A	☒
S5D3	✓	☒
S5D5	✓	☒
S5D9	✓	☒
S7G2	✓	☒

4.2.34.2 LVD HAL Module APIs Overview

The LVD HAL module defines APIs for opening, closing, statusGet and statusClear. The following table includes a complete list of the available APIs, an example API call and a short description of each API. A table of status return values follows the API summary table.

LVD HAL Module API Summary

Function Name	Example API Call and Description
open	g_lvd.p_api->open(g_lvd.p_ctrl, g_lvd.p_cfg; Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.

StatusGet	g_lvd.p_api->statusGet(g_lvd.p_ctrl, &monitor_status); Get the current state of the monitor, (threshold crossing detected, voltage currently within range) Can be used to poll the state of the LVD monitor at any time. Must be used if the peripheral was initialized with the lvd_response_tset to LVD_RESPONSE_NONE.
StatusClear	g_lvd.p_api->statusClear(g_lvd.p_ctrl); Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.
close	g_lvd.p_api->close(g_lvd.p_ctrl, g_lvd.p_cfg); Disables the LVD peripheral. Closes the driver instance.
versionGet	g_lvd.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_IN_USE	Driver already open or unable to acquire hardware lock.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	Invalid configuration value.
SSP_ERR_INVALID_MODE	If the attempted mode is invalid for this configuration.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.34.3 LVD HAL Module Operational Overview

The LVD HAL module supports the configuration and operation of the LVD monitors in the Synergy MCUs. The LVD HAL module provides configuration structures that provide all the information needed to fully configure a single LVD monitor. One instance of the LVD HAL module is needed per instance of an LVD monitor, with the exception of the LVD0 monitor. The LVD0 monitor is not configurable at runtime and must be configured at compile time via the OFS1 register.

The LVD1 and LVD2 monitors are both configurable at runtime and are configured by this module. The lvd_api_t::open function allows the user to configure and enable an LVD monitor with a single function call; the lvd_api_t::close function disables the LVD monitor. The lvd_api_t::statusGet function

returns the current status of the LVD monitor. The lvd_api_t::statusGet function should be used if the module is in polling mode, that is, without the LVD monitor interrupt enabled.

The monitor status consists of two flags. The first flag is a latched flag called lvd_status_t::crossing_detected, which indicates if the voltage being monitored has crossed the voltage threshold. In polling mode, this flag must be cleared via a call to lvd_api_t::statusClear. The flag does not need to be cleared explicitly if the LVD interrupt is in use, it will be cleared in the LVD interrupt by the driver code after the user-callback function is called. The second flag, lvd_status_t::current_state, is the instantaneous status of the monitored voltage with respect to the voltage threshold; this flag is not latched and will change as the monitored voltage changes.

The LVD HAL module can be configured to enable one or several of the LVD peripheral interrupts. If an interrupt is to be used, the user should provide a callback function for that monitor. Separate callback routines should be provided for each LVD monitor.

The LVD HAL module requires functionality provided by the BSP; this driver makes use of hardware locks provided by the BSP for register locks as well as enabling and clearing interrupts.

LVD HAL Module Important Operational Notes and Limitations

LVD HAL Module Operational Notes

- Once the appropriate values are chosen for these settings, you should add the code to call the LVD HAL module lvd_api_t::open API function to your project. This function should be called once early in the application.
- The module can be closed and reopened whenever the configuration of an LVD monitor needs to be changed. Calling the LVD lvd_api_t::open API function configures and enables the LVD hardware peripheral for the specified LVD monitor.
- The close function disables the LVD monitor and closes the driver.
- Using this module to configure the LVD peripheral to generate an interrupt requires the corresponding interrupt to be enabled in the module properties tab.
- A callback function is not required when using the LVD interrupts, but is recommended.
- A unique callback function for each LVD interrupt is not required, but is recommended.
- Clock system initialization, configuration, and runtime modification are handled outside this module. This driver only makes changes to the digital filter sample clock based on the user's choice of sample clock divisor. The digital filter sample clock is derived from the LOCO system clock.
- Not all voltage thresholds are available on all MCUs.
- Digital filtering of the VCC input to the LVD monitor is not available on all MCUs.
- The LVD driver requires functionality provided by the BSP; it makes use of hardware locks provided by the BSP for register locks as well as enabling and clearing interrupts.

LVD HAL Module Limitations

- The process of configuring and enabling a Low Voltage Detection monitor has very specific timing constraints and register write ordering. Because of these constraints, the entire process of configuring and enabling a voltage monitor is most effectively performed by a single function. The open API function performs configuration and enables the monitor in order to properly enforce the timing and register write ordering constraints.
- All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD, and CGC HOCO.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.34.4 Including the LVD HAL Module in an Application

This section describes how to include the LVD HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the LVD Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the LVD Driver is g_lvd0. This name can be changed in the associated Properties window.)

LVD HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_lvd Low Voltage Detection Driver on r_lvd	Threads	**New Stack> Driver> Power> Low Voltage Detection Driver on r_lvd**

When the LVD Driver on r_lvd is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

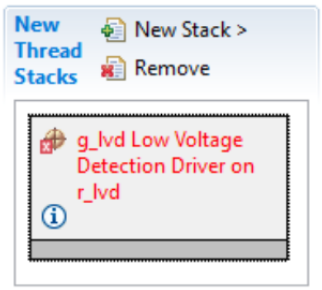


Figure 349: LVD HAL Module Stack

4.2.34.5 Configuring the LVD HAL Module

The LVD HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward

the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt priority levels.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the LVD HAL Module on r_lvd

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lvd	Module name.
Monitor number	1	Monitor number selection.
Digital filter, enable by selecting a valid sample clock rate (S7G2 only).	Digital filter is disabled, Digital filer is enabled (sampling clock is LOCO/2), Digital filer is enabled (sampling clock is LOCO/4), Digital filer is enabled (sampling clock is LOCO/8), Digital filer is enabled (sampling clock is LOCO/16) Default: Digital filter is disabled	Digital filter selection.
Voltage Threshold	Default: 2.85V (Vdet1_13)(S7G2 only).	Voltage threshold selection.
Detection Response, either reset, interrupt, non-maskable interrupt, or no response (polling mode).	Maskable interrupt triggered when voltage crosses the detection threshold, Non-maskable interrupt triggered when voltage crosses the detection threshold, Reset MCU when voltage falls below the detection threshold, No response driver will be in polled mode (using statusGet and statusClear functions). Default: Maskable interrupt triggered when voltage crosses the detection threshold	Detection response selection.

Voltage slope, rising or falling or both	Threshold crossing detected with decreasing voltage, Threshold crossing with increasing voltage, Threshold crossing detected with increasing or decreasing voltage Default: Threshold crossing detected with decreasing voltage	Indicates the direction of voltage change detection in relation to the threshold.
Negation of the monitor signal can be either be delayed from the reset event or from voltage returning to normal range	Negation of reset signal is based on delay from reset, Negation of reset signal is based on delay from voltage returning to normal range Default: Negation of reset signal is based on delay from reset	Negation of the monitor signal selection.
Monitor Interrupt Callback	NULL	Monitor interrupt callback selection.
LVD Monitor Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	LVD monitor interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

LVD HAL Module Clock Configuration

Clock system clock initialization, configuration, and runtime modification are handled outside this module. This module only makes changes to the digital filter sample clock based on the user's choice of sample clock divisor. The digital filter sample clock is derived from the LOCO system clock.

LVD HAL Module Pin Configuration

The LVD HAL module measures the voltage on the VCC pin only and does not need to be configured.

4.2.34.6 Using the LVD HAL Module in an Application

The typical steps in using the LVD HAL module in an application are:

1. Initialize the LVD HAL module using the lvd_api_t::open API.
2. If using software polling, monitor the LVD status flags with the lvd_api_t::statusGet API and process accordingly. If using the interrupt mode, process accordingly within the callback function which will return both the monitor number as well as the status.
3. Process as needed.
4. Close the LVD Instance with the lvd_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

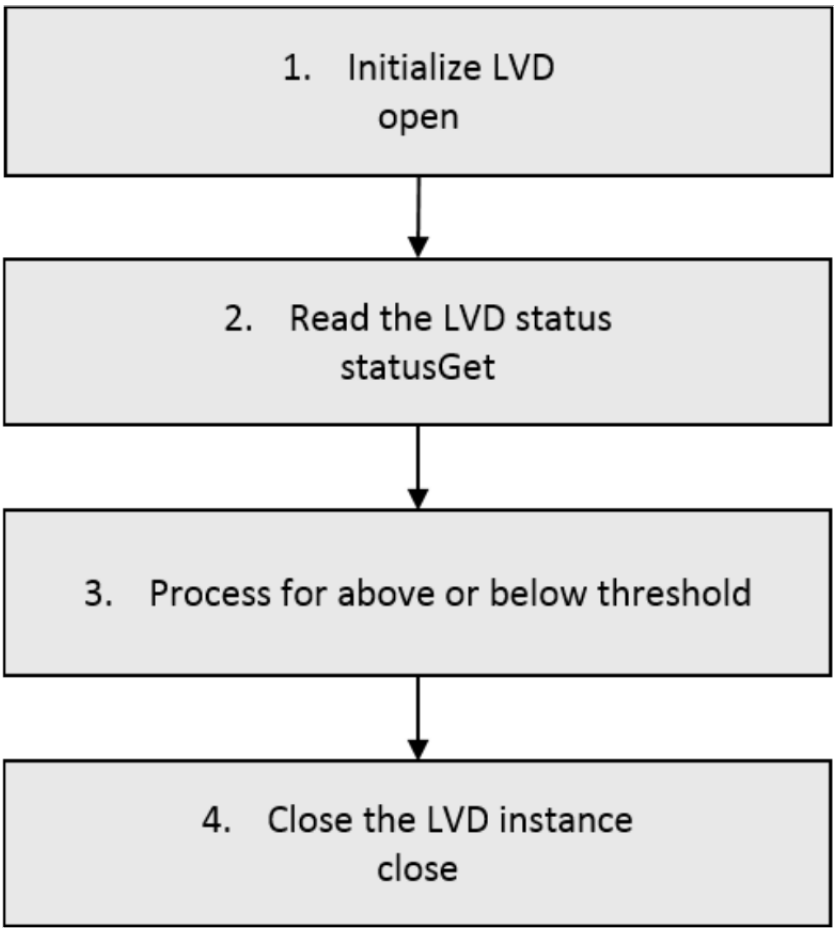


Figure 350: Flow Diagram of a Typical LVD HAL Module Application

4.2.35 OPAMP Driver

4.2.35.1 OPAMP HAL Module Introduction

The OPAMP HAL module provides a high level API for signal amplification applications and supports the OPAMP peripheral available on Synergy MCUs.

OPAMP HAL Module Features

- Low power or high-speed mode
- Start by software or AGT compare match
- Stop by software or ADC conversion end (stop by ADC conversion end only supported on op-amp channels configured to start by AGT compare match)
- Trimming available on some MCUs (see hardware manual)

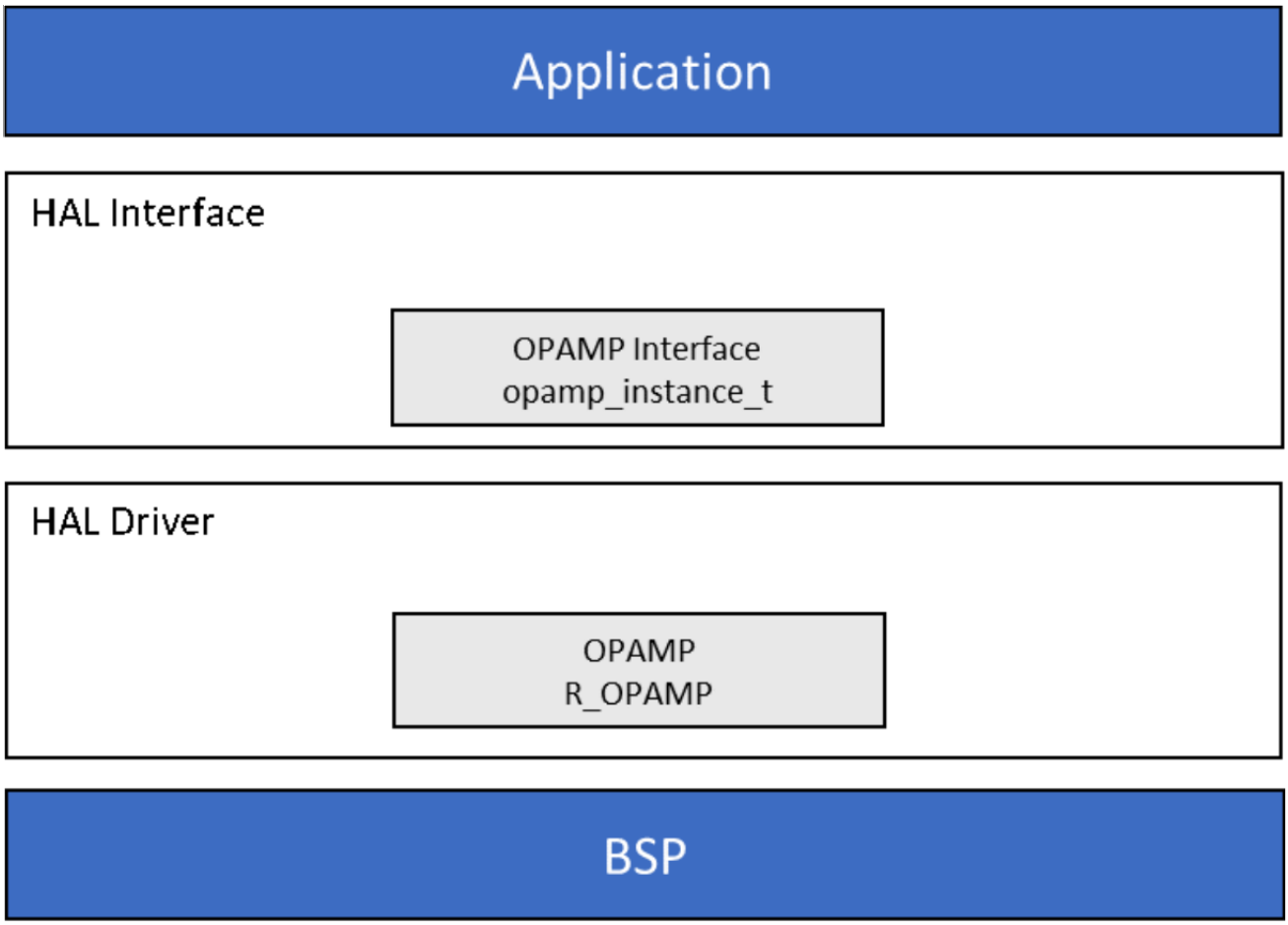


Figure 351: OPAMP HAL Module Block Diagram

OPAMP Hardware Support Details

The following hardware features are, or are not, supported by SSP for OPAMP:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Low power mode	High speed mode	Start by SW	AGT compare match	Stop by SW	ADC conversion end
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓

S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A	N/A	N/A

4.2.35.2 OPAMP HAL Module APIs Overview

The OPAMP HAL module defines API functions to open, start, stop, read status, trim and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

OPAMP HAL Module API Summary

Function Name	Example API Call and Description
open	g_opamp0.p_api->open(g_opamp0.p_ctrl, g_opamp0.p_cfg); Applies power to the OPAMP and initializes the hardware based on the user configuration.
start	g_opamp0.p_api->start(g_opamp0.p_ctrl, channel_mask); If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp.
stop	g_opamp0.p_api->stop(g_opamp0.p_ctrl, channel_mask); Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers.
trim	g_opamp0.p_api->trim(g_opamp0.p_ctrl, cmd, p_args); On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values. See Operational Notes for important details.
infoGet	g_opamp0.p_api->infoGet(g_opamp0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds.

statusGet	g_opamp0.p_api->statusGet(g_opamp0.p_ctrl, p_status); Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed.
close	g_opamp0.p_api-> close(g_opamp0.p_ctrl); Stops the op-amps.
versionGet	g_opamp0.p_api->versionGet(&version); Retrieve the module version using the version pointer

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_INVALID_STATE	The command is not valid for the current state of the trim function.
SSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.35.3 OPAMP HAL Module Operational Overview

The OPAMP HAL module controls the OPAMP peripheral on a Synergy microcontroller. It directly controls the OPAMP hardware without using any RTOS elements and provides convenient APIs to simplify development.

OPAMP HAL Module Important Operational Notes and Limitations

OPAMP HAL Module Operational Notes

Trimming the OPAMP

On MCUs that support trimming, the op-amp trim register is set to the factory default after the opamp_api_t::open API is called).

This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Not supported on all MCUs. See hardware manual for details. Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling the opamp_api_t::trim API with the command OPAMP_TRIM_CMD_START again.

The trim procedure works as follows:

- Call the opamp_api_t::trim API for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Connect a fixed voltage to the Pch (+) input.
- Connect the Nch (-) input to the op-amp output to create a voltage follower.
- Ensure the op-amp is operating and stabilized.
- Call the opamp_api_t::trim API for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
    - Call the opamp_api_t::trim API for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
    - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
    - If A <= B, call the opamp_api_t::trim API for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
- Call the opamp_api_t::trim API for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
    - Call the opamp_api_t::trim API for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
    - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
    - If A <= B, call the opamp_api_t::trim API for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

The following status return values are associated with the opamp_api_t::trim API:

SSP_SUCCESS	Conversion result in p_data
SSP_ERR_UNSUPPORTED	Trimming is not supported on this MCU.
SSP_ERR_INVALID_STATE	The command is not valid in the current state of the trim state machine.
SSP_ERR_INVALID_ARGUMENT	The requested channel is not operating or the trim procedure is not in progress for this channel/input combination.
SSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open. Trimming is not supported on all MCUs.

OPAMP HAL Module Limitations

This module only works for selected Synergy MCUs. Refer to the release notes for your current SSP release to see which MCUs are supported by this module. Additionally, the MCU Hardware Manual shows which peripherals are available.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.35.4 Including the OPAMP HAL Module in an Application

This section describes how to include the OPAMP HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the OPAMP Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the OPAMP Driver is g_opamp0. This name can be changed in the associated Properties window.)

OPAMP HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_opamp0 OPAMP Driver on r_opamp	Threads	**New Stack> Driver> Analog> OPAMP Driver on r_opamp**

When the OPAMP Driver on r_opamp is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

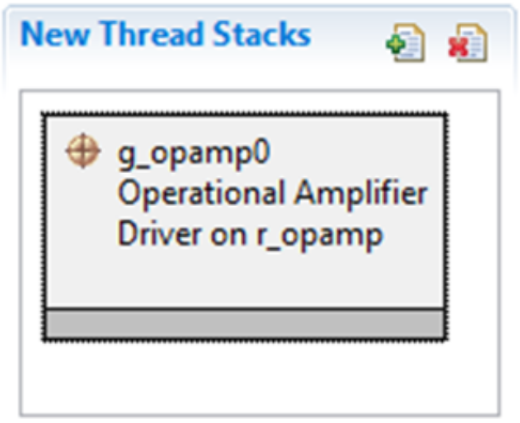


Figure 352: OPAMP HAL Module Stack

4.2.35.5 Configuring the OPAMP HAL Module

The OPAMP HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the OPAMP HAL Module on r_opamp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_opamp	Module name.
AGT Start Trigger Configuration (N/A unless AGT Start Trigger is Selected for the Channel)	AGT1 Compare Match Starts OPAMPs 0 and 2 if configured for AGT Start/AGT0 Compare Match Starts OMPAMPs 1 and 3 if configured for AGT Start, AGT1 Compare Match Starts OPAMPs 0 and 1 if configured for AGT Start/AGT0 Compare Match Starts OPAMPs 2 and 3 if configured for AGT Start, AGT1 Compare Match Starts all OPAMPs configured for AGT Start Default: AGT1 Compare Match Starts all OPAMPs configured for AGT Start	Configure which AGT channel event triggers op-amp channel. The AGT compare match event only starts the op-amp channel if the AGT Start trigger is selected in the Trigger configuration for the channel.
Power Mode	Low Power, Middle Speed, High Speed Default: High Speed	Configure the op-amp based on power or speed requirements. This setting affects the minimum required stabilization time. Middle speed is not available for all MCUs.

Trigger Channel 0	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 0. If the event trigger is selected for start, the opamp_api_t::start API enables the event trigger for this channel. If the event trigger is selected for stop, the opamp_api_t::stop API disables the event trigger for this channel.
Trigger Channel 1	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 1. If the event trigger is selected for start, the opamp_api_t::start API enables the event trigger for this channel. If the event trigger is selected for stop, the opamp_api_t::stop API disables the event trigger for this channel.
Trigger Channel 2	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 2. If the event trigger is selected for start, the opamp_api_t::start API enables the event trigger for this channel. If the event trigger is selected for stop, the opamp_api_t::stop API disables the event trigger for this channel.
Trigger Channel 3	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 3. If the event trigger is selected for start, the opamp_api_t::start API enables the event trigger for this channel. If the event trigger is selected for stop, the opamp_api_t::stop API disables the event trigger for this channel.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

OPAMP HAL Module Clock Configuration

The OPAMP HAL module does not require a specific clock configuration.

OPAMP HAL Module Pin Configuration

To use the OPAMP HAL module, the port pins for the channels receiving the analog input must be set

as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the OPAMP HAL Module on r_opamp

Resource	ISDE Tab	Pin selection Sequence
OPAMP	Pins	Select **Peripherals**> **Analog: OPAMP 0/1/2**

4.2.35.6 Using the OPAMP HAL Module in an Application

The typical steps in using the OPAMP HAL module in an application are:

1. Initialize the OPAMP module using the opamp_api_t::open API.
   Note
   Before starting any op-amp, consult the hardware manual to determine if the MCU used has internal connections switches in the OPAMP peripheral. If the OPAMP peripheral does have internal connection switches, configure the internal connections by setting the AMPnMS, AMPnPS, and AMP0OS registers directly.
2. Start the OPAMP channel(s) using the desired trigger with the opamp_api_t::start API.
   Note
   If the AGT compare match start is used, this call enables the OPAMP to be triggered by the AGT compare match. If a software trigger is used, then this call starts the OPAMP channel(s).
3. Wait for the OPAMP to stabilize. The stabilization time can be found in the hardware manual or by using the opamp_api_t::infoGet API.
4. Stop the OPAMP channel(s) by calling the opamp_api_t::stop API. (Optional)
   Note
   This stops the OPAMP regardless of if ADC conversion end triggers are enabled to stop the OPAMP.
5. Close the module and power down the peripheral using the opamp_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

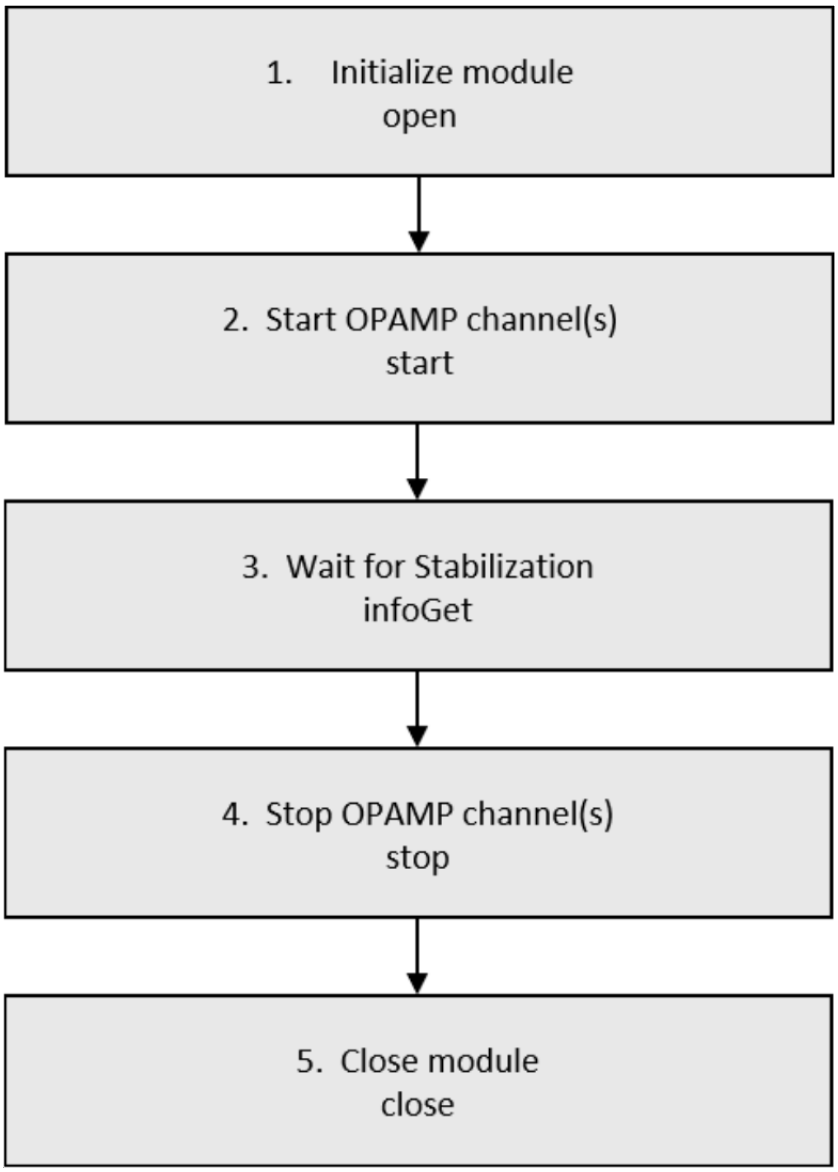


Figure 353: Flow Diagram of a Typical OPAMP HAL Module Application

4.2.36 PDC Driver

4.2.36.1 PDC HAL Module Introduction

The Parallel Data Capture Unit (PDC) HAL module provides a high-level API to capture images from a camera application and uses the PDC peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU when a capture has been completed.

PDC HAL Module Features

- Supports capture from a connected and configured camera.
- Supports a callback that informs the CPU when a capture is complete.

- Provides a pointer to the capture buffer.
- Provides an indication of the event triggering the callback.

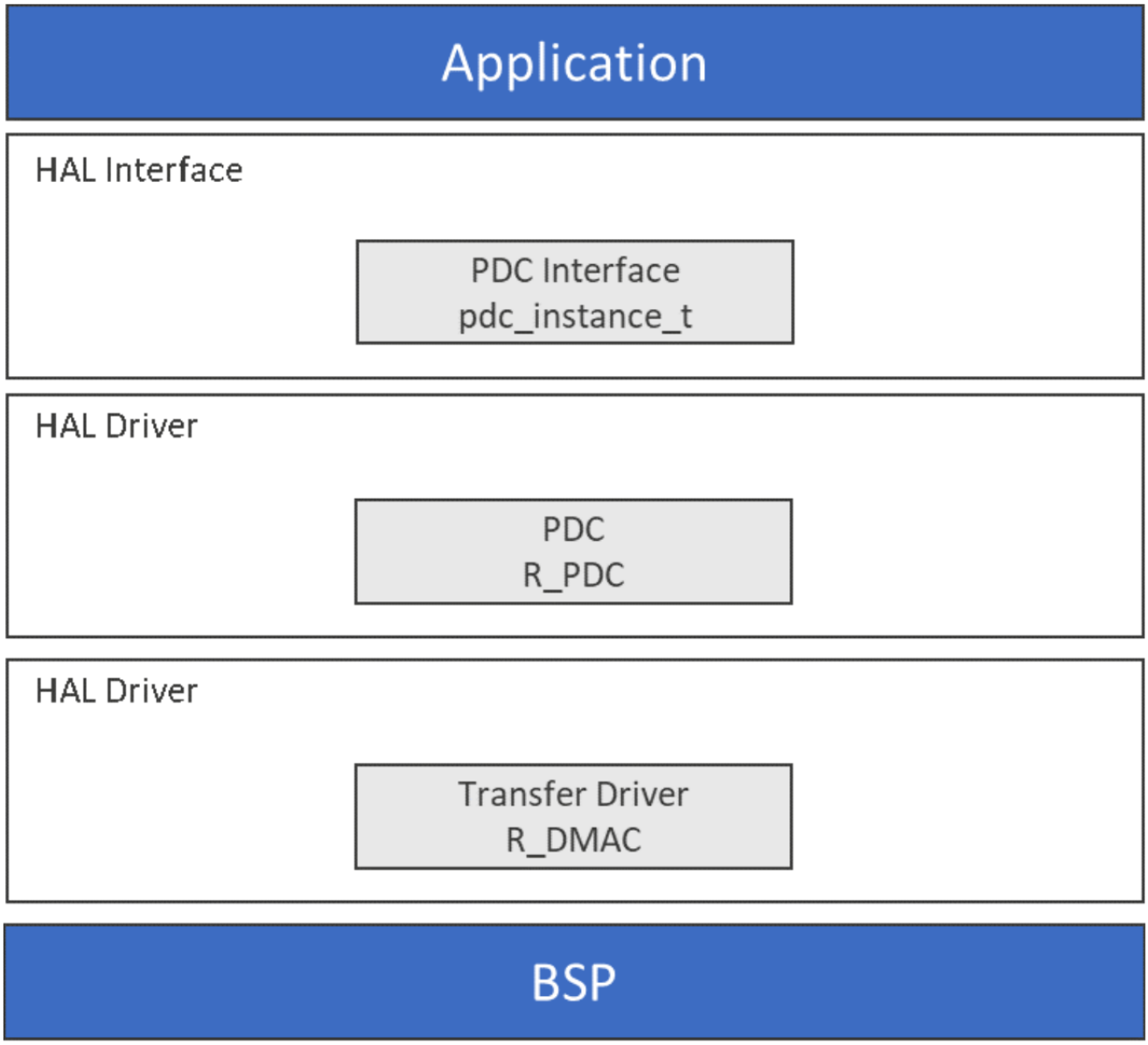


Figure 354: PDC HAL Module Block Diagram

PDC Hardware Support dDetails

The following hardware features are, or are not, supported by SSP for PDC:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Supports up to 4095 lines vertical	Supports 4 to 4095 bytes horizontal	Accepts interrupts from Receive Data Ready	Accepts interrupts from Frame End	Accepts interrupts from Overrun	Accepts interrupts from Underrun
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Accepts interrupts from Error in wrong number of lines	Accepts interrupts from Error in wrong number of bytes per line	Frame end and receive data ready interrupts can start DTC	Frame end and receive data ready interrupts can start DMAC	Frequency division ratio: Selectable from 2, 4, 6, 8, 10, 12, 14, and 16	Supports PDC Reset Function
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	✓	✓	☒	☒	☒	☒
S5D9	✓	✓	☒	☒	☒	☒
S7G2	✓	✓	☒	☒	☒	☒

MCU Group	Supports Selectable active polarity for VSYNC and HSYNC signals	Supports Monitoring of VSYNC and HSYNC signals	Endian order selectable
S124	N/A	N/A	N/A
S128	N/A	N/A	N/A
S1JA	N/A	N/A	N/A
S3A1	N/A	N/A	N/A
S3A3	N/A	N/A	N/A
S3A6	N/A	N/A	N/A
S3A7	N/A	N/A	N/A
S5D3	N/A	N/A	N/A
S5D5	✓	✓	✓
S5D9	✓	✓	✓
S7G2	✓	✓	✓

4.2.36.2 PDC HAL Module APIs Overview

The PDC HAL module defines APIs for opening, closing and starting data capture. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

PDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_pdc.p_api->open(g_pdc.p_ctrl, g_pdc.p_cfg) Initial configuration.
close	g_pdc.p_api->close(g_pdc.p_ctrl)) Closes the driver and allows reconfiguration. May reduce power consumption.
captureStart	g_pdc.p_api->captureStart(g_pdc.p_ctrl, NULL) Start a capture.
stateGet	g_pdc.p_api->stateGet(g_pdc.p_ctrl, &state_data) Get the state of VSYNC and HSYNC pins.
versionGet	g_pdc.p_api->versionGet(&version) Return the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ALREADY_OPEN	Unit is already open.
SSP_ERR_HW_LOCKED	Unable to reserve BSP hardware lock.
SSP_ERR_TIMEOUT	Reset Operation timed out.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.36.3 PDC HAL Module Operational Overview

The capture operation requires a configured external camera connected to the Synergy microcontroller. Before performing a capture, the camera must be configured and it must generate a PIXCLK-clock input into the microcontroller. In some instances, a camera requires a running-clock input before it can be configured.

Use the call pdc_api_t::open API (which configures and starts the PCKO-clock output from the PDC into the camera) before initializing the camera. Once the camera is configured, the pdc_api_t::captureStart can be called to capture an image. Configuration of a camera module may require the use of an I^2C or SPI interface.

PDC HAL Module Important Operational Notes and Limitations

PDC HAL Module Operational Notes

In most instances, the data rate from a camera or the PDC peripheral is too fast to be serviced by the CPU in an interrupt service routine (ISR). Therefore, this module requires an implementation of the transfer driver on the DMAC to perform a high-speed transfer from the PDC peripheral and memory.

Both the PDC frame-end and PDC error interrupts must be used to generate interrupts in the following situations:

- An interrupt when an image is captured (frame end)
- An interrupt when an error occurs

Data Buffer Setting

If p_buffer is set to anything other than NULL, one or more data buffers are created to store image data. The size of each buffer is calculated using the following formula:

Buffer size (bytes) = x_capture_pixels x y_capture_pixels x bytes_per_pixel

For large resolution cameras, the captured image could result in a large amount of data. It may be necessary to locate the buffer(s) in external memory (such as, SDRAM). Consideration should be

given to bus bandwidth when using external memory.

For example, when using a high frame-rate camera to do an image capture via the PDC into SDRAM, and using SDRAM to hold the display buffer for an LCD display with a high refresh rate, may cause a data bottleneck from the PDC to memory that results in an overrun-error condition.

Note

If p_buffer is set to NULL, no memory is allocated to store the captured image data. The application must ensure that there is suitable memory of sufficient size available to the PDC. The PDC could capture directly into the display buffer of a connected LCD panel.

PDC HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.36.4 Including the PDC HAL Module in an Application

This section describes how to include the PDC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the PDC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the PDC Driver is g_pdc0. This name can be changed in the associated Properties window.)

PDC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_pdc0 PDCDriver on r_pdc	Threads	**New Stack> Driver> Graphics> PDC Driver on r_pdc**

When the PDC Driver on r_pdc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

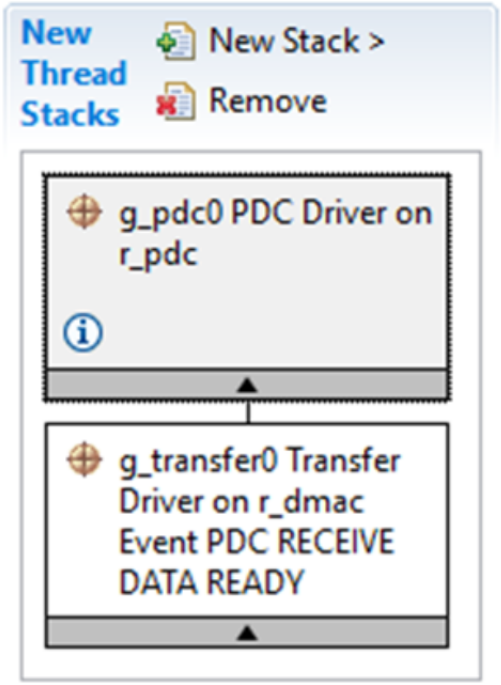


Figure 355: PDC HAL Module Stack

4.2.36.5 Configuring the PDC HAL Module

The PDC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the PDC HAL Module on r_pdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_pdc0	The name of the PDC module instance. Specify arbitrary C symbol.

Name of the data buffer to store image data	g_user_buffer	Specify the name of the data buffer to create or set to NULL if it is to be created by the user external to the PDC driver.
Section where data buffer is allocated	sdram	Specify the RAM section for the image data buffer. Typically bss (internal RAM) or sdram.
Number of bytes per pixel	2	Specify the number of bytes per pixel of the captured image data.
Number of image data buffers	1	Specify the number of buffers to create.
Clock Divider	CLK/2, CLK/4, CLK/6, CLK/8, CLK10, CLK12, CLK14, CLK16 Default: CLK/2	Specify the clock divider of the clock input to the PDC peripheral.
Endian of image data	Little, Big Default: Little	Specify the endian of the captured image data.
HYSNC signal polarity	High, Low Default: High	Specify the active polarity of the HSYNC signal.
VSYNC signal polarity	High, Low Default: High	Specify the active polarity of the VSYNC signal.
Number of pixels to capture horizontally	640	Number of horizontal pixels to capture.
Number of pixels to capture vertically	480	Number of vertical lines to capture.
Horizontal pixel to start capture from	0	Horizontal pixel to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Line to start capture from	0	Vertical line to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.

Callback	g_pdc_user_callback	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time a frame is captured and ready to be processed. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Frame End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	The driver needs a valid interrupt priority setting. It will not function if disabled.
PDC Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	The driver needs a valid interrupt priority setting. It will not function if disabled.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the PDC HAL Module Lower-Lever Module

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the Red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dmac Event PDC RECEIVE DATA READY

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_transfer0	Driver name.
Mode	Block	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.

Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	8	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection.
Activation Source (Must enable IRQ)	Event PDC RECEIVE DATA READY	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

PDC HAL Module Clock Configuration

The PDC uses the PCLKB as its clock source. The only restriction when configuring this clock is that the PIXCLCK should be less than 0.6 x PCLKB so the PCLKB frequency should be set accordingly.

PDC HAL Module Pin Configuration

The PDC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table depicts the method to select pins within the SSP configuration window and the subsequent table gives an example selection for PDC pins:

Pin Selection for the PDC HAL Module on r_pdc

Resource	ISDE Tab	Pin selection Sequence
PDC	Pins	Select **Peripherals**> **Graphics: PDC**> **PDC0**

Note

The selection sequence assumes KINT0 is the desired hardware target for the driver.

Pin Configuration Settings for the PDC HAL Module on r_pdc

Property	Value	Description

Pin Group Selection	Mixed, _A Only Default: Mixed	Pin group selection.
Operation Mode	Disabled, Custom, Enabled Default: Disabled	Select Enabled as the Operation Mode for PDC.
HSYNC	None, P704 Default: None	HSYNC Pin.
PCKO	None, P511 Default: P511	PCKO Pin.
PIXCLK	None, P705 Default: None	PIXCLK Pin.
VSYNC	None, P512 Default: P512	VSNC Pin.
PIXD0:7	None, Pnnn Default: None	PIX Data0:7 Pins.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.36.6 Using the PDC HAL Module in an Application

The typical steps in using the PDC HAL module in an application are:

1. Initialize the PDC HAL module using the pdc_api_t::open API.
2. Configure the camera as needed.
3. Start image capture using the pdc_api_t::captureStart API.
4. Callback is called when image is captured.
5. Read state of HSYNC and VSTNC using pdc_api_t::stateGet API.
6. Process data as needed.
7. Close using the pdc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

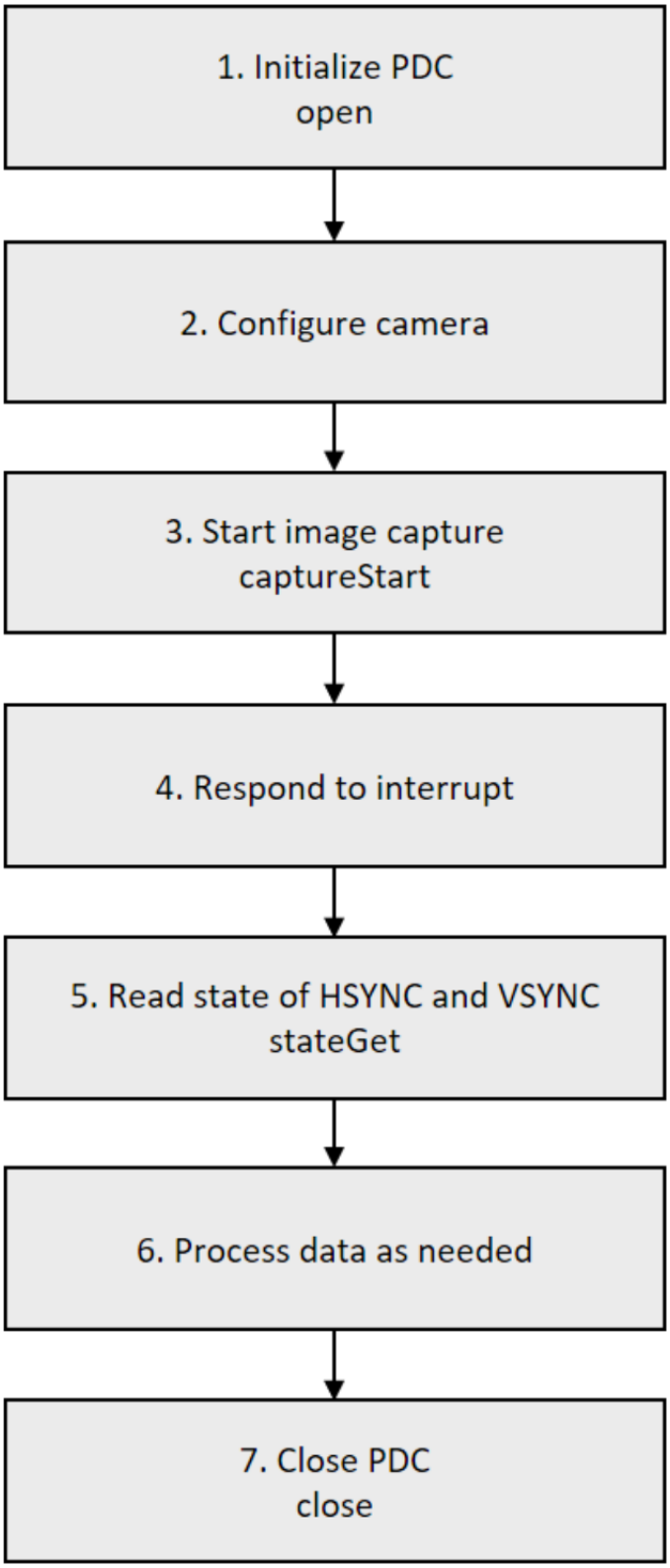


Figure 356: Flow Diagram of a Typical PDC HAL Module Application

4.2.37 PTP Driver on r_ptp

4.2.37.1 Precision Time Protocol HAL Module Introduction

The PTP HAL Module provides high-level APIs for the time synchronization using the PTP function of the EPTPC peripheral module (EPTPC) on the synergy MCU. A user callback can be created to indicate the occurrence of pulse output timer event or the state change of Synchronous Frame Processing units(SYNFP0 and SYNFP1), Statistical Time Correction Algorithm(STCA), or Packet Relation Controller unit(PRC-TC).

Precision Time Protocol HAL Module Features

The PTP HAL module configures the PTP for time synchronization functionality.

The PTP HAL allows the user to perform the following operations:

- Set and get local clock counter value.
- Set and get master port ID.
- Set and get PTP message reception configuration.
- Update PTP message information.
- Get offset from master and mean path delay values.
- Collecting gradient differences and extracting the worst ten values by hardware and software.
- Set start time, period, and pulse width of pulse output timer.
- Indicate and auto-clear pulse output event signals to the ELC.
- Enable and disable INFABT status notification.

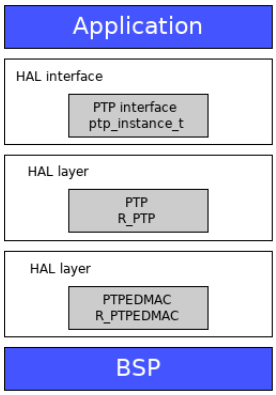


Figure 357: PTP HAL Module Block Diagram

Precision Time Protocol Hardware Support Details

The following hardware features are, or are not, supported by SSP for PTP.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Ordinary clock	Boundary clock	Transparent clock	E2E delay mechanism	P2P delay mechanism	Pulse output timer operation
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	N/A	N/A	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	✓	N/A	N/A	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

4.2.37.2 Precision Time Protocol HAL Module APIs Overview

The PTP HAL module interface defines APIs for key features such as initializing, starting, stopping, getting and setting of synchronous information, setting start time for pulse output timer, setting ELC event indication, and updating PTP message flags. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

PTP HAL Module API Summary

PTP API functions

Function Name	Example API Call and Description
open	g_ptp.p_api->open(g_ptp.p_ctrl, g_ptp.p_cfg); Open the PTP driver module.
close	g_ptp.p_api->close(g_ptp.p_ctrl); Close the PTP driver module.
configure	g_ptp.p_api->configure(g_ptp.p_ctrl, ip_address, physical_address_msw, physical_address_lsw); Configures the PTP driver module.
setExtPromiscuous	g_ptp.p_api->setExtPromiscuous(g_ptp.p_ctrl, 1, false); Sets or clears the extended promiscuous mode
setLocalClock	g_ptp.p_api->setLocalClock(g_ptp.p_ctrl, &lc_clk); Sets local clock counter

getLocalClock	g_ptp.p_api->getLocalClock(g_ptp.p_ctrl, &lc_clk, 20); Gets local clock counter.
getMasterPortID	g_ptp.p_api->getMasterPortID(g_ptp.p_ctrl, 1, curClkId, &curPortId); Gets master port ID.
setMasterPortID	g_ptp.p_api->setMasterPortID(g_ptp.p_ctrl, 1, curClkId, &curPortId); Sets master port ID.
getSyncInfo	g_ptp.p_api->getSyncInfo(g_ptp.p_ctrl, 1, &master_offset, &path_delay); Get current offsetFromMaster and meanPathDelay.
start	g_ptp.p_api->start(g_ptp.p_ctrl, 100); Starts the time synchronization.
stop	g_ptp.p_api->stop(g_ptp.p_ctrl, 100); Stops the time synchronization.
checkWorst10Values	g_ptp.p_api->checkWorst10Values(g_ptp.p_ctrl, 40000); Checks worst 10 values acquired by hardware and set as gradient limits.
setWorst10Values	g_ptp.p_api->setWorst10Values(g_ptp.p_ctrl, 32); Sets the time interval for collecting worst 10 values.
getWorst10Values	g_ptp.p_api->getWorst10Values(g_ptp.p_ctrl, p_lim, m_lim, 40000); Gets the worst 10 values acquired by software.
setGradientLimit	g_ptp.p_api->setGradientLimit(g_ptp.p_ctrl, p_lim, m_lim); Sets the gradient limits for positive and negative worst 10 values.
updateClockID	g_ptp.p_api->updateClockID(g_ptp.p_ctrl, 1, clock_id); Updates clock identity field.
updateDomainNumber	g_ptp.p_api->updateDomainNumber(g_ptp.p_ctrl, 1, domain_num); Updates domain number field in the message header.
updateAnnounceFlags	g_ptp.p_api->updateAnnounceFlags(g_ptp.p_ctrl, 1, &p_flag); Updates announce message's flag field.
updateAnnounceMsgs	g_ptp.p_api->updateAnnounceMsgs(g_ptp.p_ctrl, 1, &p_message); Updates announce message's message field.

updateSyncAnnounceMsgInterval	g_ptp.p_api->updateSyncAnnounceMsgInterval(g_ptp.p_ctrl, 1, &p_sync_interval, &p_ance_interval); Updates transmission interval and logMessageInterval of Sync and Announce messages.
updateDelayMsgInterval	g_ptp.p_api->updateDelayMsgInterval(g_ptp.p_ctrl, 1, &p_interval, &p_timeout); Updates transmission interval, logMessageInterval and timeout values of Delay message.
getMessageReceptionConfig	g_ptp.p_api->getMessageReceptionConfig(g_ptp.p_ctrl, 1, &p_ptp_message_reception); Gets PTP message reception synchronous configuration.
setMessageReceptionConfig	g_ptp.p_api->setMessageReceptionConfig(g_ptp.p_ctrl, PTP_TEST_CHANNEL, &p_ptp_message_reception); Sets PTP message reception synchronous configuration.
disableTimer	g_ptp.p_api->disableTimer(g_ptp.p_ctrl, PTP_STCA_TIMER_CHANNEL_0));Disables the specified timer event interrupt.
indicateEvent	g_ptp.p_api->indicateEvent(g_ptp.p_ctrl, cyc_ch, PTP_STCA_TIMER_PULSE_EDGE_RISING, true); Sets or clears ELC interrupt indication.
autoClearEvent	g_ptp.p_api->autoClearEvent(g_ptp.p_ctrl, cyc_ch, PTP_STCA_TIMER_PULSE_EDGE_RISING, true); Sets or clears ELC interrupt auto clear mode.
setTimer	g_ptp.p_api->setTimer(g_ptp.p_ctrl, 0U, start_time, 200000, 100000);Sets start time, pulse period and pulse width for the pulse output timer.
setMINTevent	g_ptp.p_api->setMINTevent(g_ptp.p_ctrl, PTP_EVENT_STCA, 0x03, false);Sets EPTPC MINT interrupt event.
enableINFABTnotification	g_ptp.p_api->enableINFABTnotification(g_ptp.p_ctrl, 1); Enables EPTPC INFABT notification of the specified PTP channel.
disableINFABTnotification	g_ptp.p_api->disableINFABTnotification(g_ptp.p_ctrl, 1); Disables EPTPC INFABT notification of the specified PTP channel.

checkINFABTstatus	g_ptp.p_api->checkINFABTstatus(g_ptp.p_ctrl, &status); Checks the status of INFABT flag of the specified PTP channel.
clearINFABTstatus	g_ptp.p_api->clearINFABTstatus(g_ptp.p_ctrl, 1); Clears INFABT interrupt occurrence flag of the specified PTP channel.
versionGet	g_ptp.p_api->versionGet(&ptp_version); Get the driver version based on compile time macros.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.
While calling configure API, make sure that IP address and MAC address are assigned to respective index when using ordinary clock 0 and ordinary clock 1.
checkWorst10Values API should be called only when STCA synchronous mode is set to Mode 2 - Gradient collection by hardware in the configurator.
getWorst10Values and setGradientLimit APIs should be called only when STCA synchronous mode is set to Mode 2 - Gradient collection by software in the configurator.
linkProcess API should be called after configure API and ensure that there is no delay between the API calls.
While calling setTimer API, user needs to pass half the required pulse output period and pulse high width.

PTP HAL Module Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_CHANNEL	Attempt to use invalid PTP channel
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_INVALID_MODE	Attempt to use unsupported or incorrect mode.
SSP_ERR_IN_USE	Unit is already opened.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.37.3 Precision Time Protocol HAL Module Operational Overview

The PTP driver on r_ptp HAL module controls the on-chip Precision Time Protocol (PTP) module for the Ethernet Controller (EPTPC) on a Synergy microcontroller, as configured by the user. It directly controls the EPTPC hardware without using any RTOS elements. It provides convenient API functions to simplify development.

Precision Time Protocol HAL Module Important Operational Notes and Limitations

Precision Time Protocol HAL Module Operational Notes

PTP HAL module can be used for the synchronization of clocks.

Clock state

- **Master**: EPTPC peripheral sends announce and sync message to slave devices which contain local clock time in UNIX Timestamp format
- **Slave**: EPTPC peripheral receives and corrects the local time in accordance to master clock time.

Clock mode

- **Ordinary clock**: PTP messages are transmitted and received through one Ethernet port in operation as an ordinary clock. An ordinary clock operates as either master or slave.
- **Boundary clock**: PTP messages are transmitted and received through both ports in operation as a boundary clock. One port operates as a slave in synchronization with the master and the other operates as the master that delivers time information synchronized with the master clock. Both ports can also operate as masters.
- **Transparent clock**: A transparent clock does not act as a master or slave, instead relays PTP messages from the master to the slave.

Delay correction mechanism

- **E2E delay correction mechanism**: It involves message exchange between master and slave. The slave sends a delay request to the master, which in turn responds with a delay response back to the requested slave.
- **P2P delay correction mechanism**: This mechanism uses a port-based peer delay message mechanism. Each port on a PTP device sends peer delay request messages to the port it is directly connected to. The connected port then responds with a peer delay response message.

Statistical time correction algorithm (STCA): When configured as a slave, the synchronized state can be identified by the offset from master value staying below a threshold specified in advance or calculated statistically from collected positive and negative gradient values by hardware or software (worst-10 acquisition).

STCA clock can be used as the clock source for generating pulse signals from pulse output timer m (m = 0 to 5). PCLKA is used as a source clock at a frequency of 20 MHz.

The generated pulse signals can be linked through event link (ELC) to get triggered when a rising or falling edge is detected.

Interrupts and callback: The user callback must be registered to handle pulse output timer event or state change of SYNFP0, SYNFP1, STCA or PRC-TC. If the user callback is registered then the PTP driver will invoke the callback (ptp_cfg_t::p_callback) with argument ptp_callback_args_t, indicating the event ptp_event_t.

PTPEDMAC supports the following interrupts:

1. Rising edge detected on the periodic pulse signal from pulse output timer
2. Change in state of SYNFP0
3. Change in state of SYNFP1

4. Change in state of STCA
5. Change in state of PRC-TC

Initialization: Ethernet initialization should be done and Ethernet link should be up before calling PTP APIs.

Precision Time Protocol HAL Module Limitations

PTP HAL module limitations:

1. PTP HAL module does not support the setting of inter-port transfer mode for standard ethernet frames.
2. PTP HAL module does not support the setting of multicast frame filter for standard ethernet frames.

EPTPC hardware limitations:

1. PTP message frames are not supported over IPv6 format.

4.2.37.4 Including the Precision Time Protocol HAL Module in an Application

This section describes how to include the PTP HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

PTP HAL driver is used as a lower level module for Ethernet and is not available to add as a separate stack. It is available as optional driver under sf_el_nx. PTPEDMAC is added as a lower level module for PTP automatically. An example is illustrated in the following figure where a module must be selected to Add PTP Driver. The following figure shows the selection of the PTP driver to complete the stack for the module.

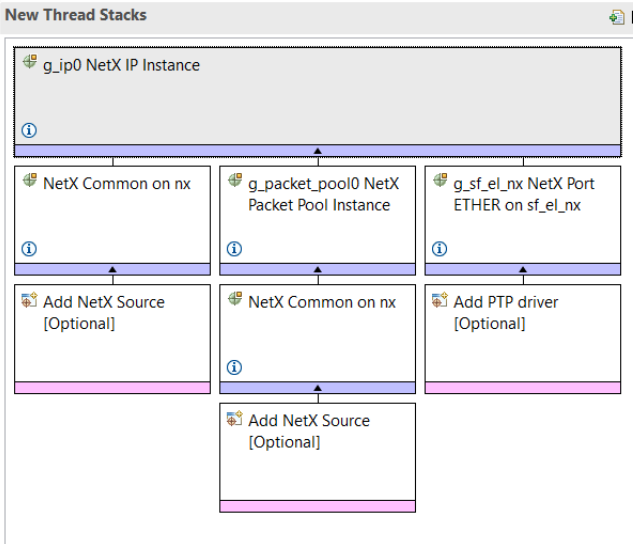


Figure 358: PTP HAL Module Stack 1

When the PTP Driver on r_ptp is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

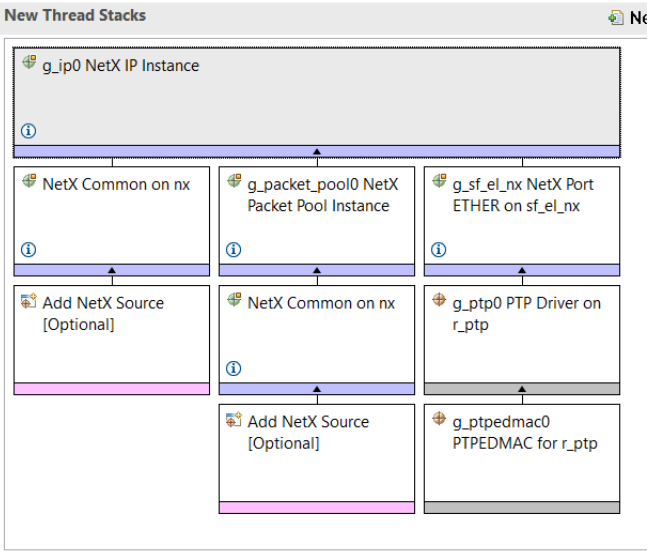


Figure 359: PTP HAL Module Stack 2

4.2.37.5 Configuring the Precision Time Protocol HAL Module

The PTP HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the PTP HAL Module on r_ptp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.

Name	g_ptp	Module name.
Clock mode	Ordinary clock 0, Ordinary clock 1, Boundary clock, Transparent clock Default: Ordinary clock 0	PTP clock mode selection.
STCA synchronous mode	Mode 1- No gradient collection, Mode 2- Gradient collection by hardware, Mode 2- Gradient collection by software Default: Mode 2- Gradient collection by hardware	STCA synchronous mode selection.
MINT callback	Default: NULL	PTP MINT interrupt callback selection.
MINT interrupt priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the MINT interrupt priority.
Clock state	Slave, Master Default: Slave	PTP clock state selection.
Delay correction mechanism	E2E (End to end), P2P (Peer to peer) Default: End to end delay correction mechanism	Delay correction mechanism selection.
Frame format	Ethernet II, Ethernet 802.3, Ethernet II over UDP4, Ethernet 802.3 over UDP4 Default: Ethernet II	PTP message frame format selection.

Note

While using ordinary clock mode, make sure the Ethernet channel selected for sf_el_nx and r_ptp are the same.

Other MCUs may have different default values and available configuration settings.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group.

Precision Time Protocol HAL Module Clock Configuration

No specific clock configurations are required for the PTP HAL Module.

Precision Time Protocol HAL Module Pin Configuration

No specific pin configurations are required for the PTP HAL Module.

4.2.37.6 Using the Precision Time Protocol HAL Module in an Application

Note

User shall proceed with time synchronization operations only after Ethernet is initialized, link is up and PTP is configured with IP and MAC address.

The typical steps in using the PTP HAL module in an application are:

PTP master application:

1. Initialize the PTP HAL module using ptp_api_t::open, ptp_api_t::configure, ptpedmac_api_t::open APIs.

2. Set PTP host interface for PTP message transfer using the ptpedmac_api_t::linkProcess API.

3. Set the local clock value using the ptp_api_t::setLocalClock API.

4. Start time synchronization using the ptp_api_t::start API.

5. Stop synchronization using the ptp_api_t::stop API.

6. Use the ptp_api_t::close call to close the peripheral.

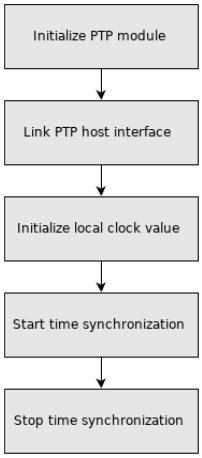


Figure 360: Flow Diagram of a PTP HAL Module Application as master

PTP slave application:

1. Initialize the PTP HAL module using ptp_api_t::open, ptp_api_t::configure, ptpedmac_api_t::open APIs.
2. Set the interval for getting worst 10 values using the ptp_api_t::setWorst10Values API.
3. Set PTP host interface for PTP message transfer using the ptpedmac_api_t::linkProcess API.
4. Wait till PINT interrupt occurs. Receive PTP message using ptpedmac_api_t::read API through ptpedmac_cfg_t::p_callback.
5. Start time synchronization using the ptp_api_t::start API.
6. Get worst 10 value acquired by software or hardware using ptp_api_t::getWorst10Values or ptp_api_t::checkWorst10Values and set as gradient limit.
7. Get local clock counter value using the ptp_api_t::getLocalClock API.
8. Get master offset and path delay using ptp_api_t::getSyncInfo API.
9. Stop time synchronization using the ptp_api_t::stop API.
10. Use the ptp_api_t::close call to close the peripheral.

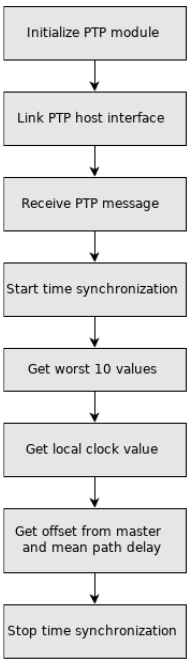


Figure 361: Flow Diagram of a PTP HAL Module Application as slave

Application for setting pulse output timer:

1. Configure PTP clock state as master and register a callback function for MINT ISR.
2. Initialize the PTP HAL module using ptp_api_t::open, ptp_api_t::configure APIs.
3. Configure PTP module to indicate and auto-clear event using ptp_api_t::indicateEvent, ptp_api_t::autoClearEvent APIs.
4. Set the start time, period, and pulse width for pulse output timer using ptp_api_t::setTimer API
5. Set the local clock value using the ptp_api_t::setLocalClock API.
6. Start time synchronization using the ptp_api_t::start API.
7. Wait till the MINT timer event occurs. Start counting by pulse output timer through ptp_cfg_t::p_callback.
8. Stop synchronization using the ptp_api_t::stop API.
9. Use the ptp_api_t::close call to close the peripheral.

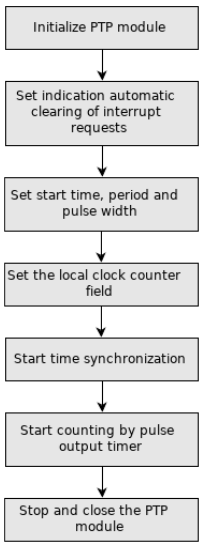


Figure 362: Flow Diagram of a PTP HAL Module Application for setting pulse output timer

Note

As per hardware user manual, time synchronization will not start immediately after starting the PTP module by calling start API. It might take about 3 seconds to get synchronized.

4.2.38 PTPEDMAC Driver on r_ptpedmac

4.2.38.1 PTPEDMAC HAL Module Introduction

The PTPEDMAC HAL Module provides high-level APIs for message transmission using a DMA controller for EPTPC peripheral in synergy MCU. A user callback can be created to indicate the occurrence of frame receive complete event, frame transmit complete event, or error event.

PTPEDMAC HAL Module Features

The PTPEDMAC HAL module configures PTPEDMAC for data transmission and reception of PTP messages.

The PTPEDMAC HAL allows the user to perform the following operations:

- Link the PTP host interface to transmit and receive PTP messages.
- Check the communication status.
- Read PTP messages.

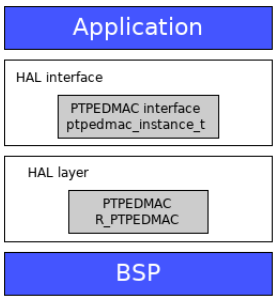


Figure 363: PTPEDMAC HAL Module Block Diagram

PTPEDMAC Hardware Support Details

The following hardware features are, or are not, supported by SSP for PTPEDMAC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Linking PTP host Interface	Read PTP message
S124	N/A	N/A
S128	N/A	N/A
S1JA	N/A	N/A
S3A1	N/A	N/A
S3A3	N/A	N/A
S3A6	N/A	N/A
S3A7	N/A	N/A
S5D3	N/A	N/A
S5D5	N/A	N/A
S5D9	✓	✓
S7G2	✓	✓

4.2.38.2 PTPEDMAC HAL Module APIs Overview

The PTPEDMAC module interface defines APIs for key features such as opening, closing, reading, linking host interface. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

PTPEDMAC HAL Module API Summary

PTPEDMAC API functions

Function Name	Example API Call and Description
open	g_ptpedmac.p_api->open(g_ptpedmac.p_ctrl, g_ptpedmac.p_cfg); Open the PTPEDMAC driver module for reception of PTP messages.
close	g_ptpedmac.p_api->close(g_ptpedmac.p_ctrl); Close the PTPEMAC module.
linkProcess	g_ptpedmac.p_api->linkProcess(g_ptpedmac.p_ctrl); Sets host interface to transfer PTP messages
linkCheck	g_ptpedmac.p_api->linkCheck(g_ptpedmac.p_ctrl); Checks host interface communication status
read	g_ptpedmac.p_api->read(g_ptpedmac.p_ctrl, &read_ch, P_BUF, &num_recvd); Receives PTP message.
versionGet	g_ptpedmac.p_api ->versionGet (&ptpedmac_version); Get the driver version based on compile time macros.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.
PTPEDMAC open API should be called after PTP open API call is successful.

PTPEDMAC HAL Module Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.
SSP_ERR_NOT_ENABLED	PTP host interface is not enabled.
SSP_ERR_TIMEOUT	Timeout error.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.38.3 PTPEDMAC HAL Module Operational Overview

The PTPEDMAC HAL module provides DMA controller for message transmission and reception for the on-chip Ethernet PTP Controller (EPTPC) on a Synergy microcontroller, as configured by the user. It provides convenient API functions to simplify development.

PTPEDMAC HAL Module Important Operational Notes and Limitations

PTPEDMAC HAL Module Operational Notes

The PTPEDMAC controls data transmission and reception based on EPTPC configuration.

The PTPEDMAC transfers data according to the information written in the descriptor. A descriptor includes the buffer size, address, and transmit or receive status.

The user callback must be registered to indicate PTPEDMAC communication status. If the user callback is registered then the PTPEDMAC driver will invoke the callback (ptpedmac_cfg_t::p_callback) with argument ptpedmac_callback_args_t, indicating the event ptpedmac_event_t.

PTPEDMAC supports the following interrupts:

1. Frame receive complete event - this event occurs when PTP message frame is received successfully

    1. Frame transmit complete event - this event occurs when the PTP message frame is transmitted successfully
    2. Error event - this event occurs when an error occurs during the transfer of PTP message

PTPEDMAC HAL Module Limitations

PTPEDMAC write functionality is not supported as EPTPC hardware automatically handles analysis and transmission of PTP messages for time synchronization.

4.2.38.4 Including the PTPEDMAC HAL Module in an Application

This section describes how to include the PTPEDMAC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

PTPEDMAC HAL driver is used as a lower-level module for PTP driver and is not available to add as a separate stack. PTPEDMAC is added as a lower-level module for PTP driver automatically.

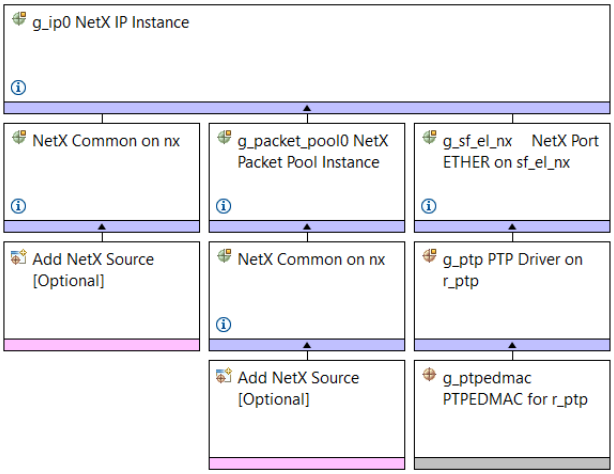


Figure 364: PTPEDMAC HAL Module Stack

When the PTPEDMAC Driver on r_ptpedmac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

4.2.38.5 Configuring the PTPEDMAC HAL Module

The PTPEDMAC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the PTPEDMAC HAL Module on r_ptpedmac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Number of RX descriptors	Range: 1 to 8 Default: 1	Number of receive buffer descriptors selection.

PTPEDMAC buffer size	Range: 64 to 1536 Default: 1536	Buffer size.
Name	g_ptpedmac0	Module name.
Callback	Default: NULL	PINT interrupt callback selection.
PINT interrupt priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Select the PINT interrupt priority.

Other MCUs may have different default values and available configuration settings.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group.

PTPEDMAC HAL Module Clock Configuration

No specific clock configurations are required for the PTPEDMAC HAL Module.

PTPEDMAC HAL Module Pin Configuration

No specific pin configurations are required for the PTPEDMAC HAL Module.

4.2.38.6 Using the PTPEDMAC HAL Module in an Application

Note

PTPEDMAC APIs are to be used along with PTP APIs for message transmission and reception.
User shall call ptpedmac_api_t::open API only after ptp_api_t::open API call is successful.

The typical steps in using the PTPEDMAC HAL module in an application are:

1. Initialize the PTP HAL module using ptp_api_t::open API.
2. Initialize the PTPEDMAC module using ptpedmac_api_t::open API.
3. Configure the PTP module with IP and MAC address using ptp_api_t::configure API.
4. Link PTPEDMAC to receive and transfer PTP message transfer using the ptpedmac_api_t::linkProcess API.
5. Start time synchronization using the ptp_api_t::start API.
6. Receive PTP message using ptpedmac_api_t::read API through ptpedmac_cfg_t::p_callback.

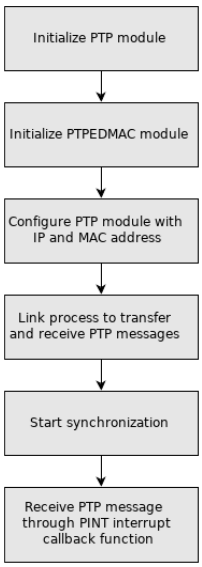


Figure 365: Flow Diagram of a PTPEDMAC HAL Module Application

4.2.39 QSPI Driver

4.2.39.1 QSPI HAL Module Introduction

The Quad SPI (QSPI) HAL module provides a high-level API for erasing and programming the contents of a QSPI flash device connected to the microcontroller. Unlike many other modules, there is no callback function for the QSPI.

QSPI HAL Module Features

The QSPI HAL Module is used to initialize the QSPI peripheral that allows erasing and programming the contents of a QSPI flash device connected to the microcontroller over the Quad SPI interface. Key features include:

- Accessing Quad SPI flash devices using Direct Communication Mode
- Reading data from a QSPI flash device
- Programming the page of a QSPI flash device
- Erasing sectors of a QSPI flash device
- Selecting a bank to control access to a QSPI flash device
- Accessing Quad SPI flash devices using 3-byte/4-byte Addressing Mode

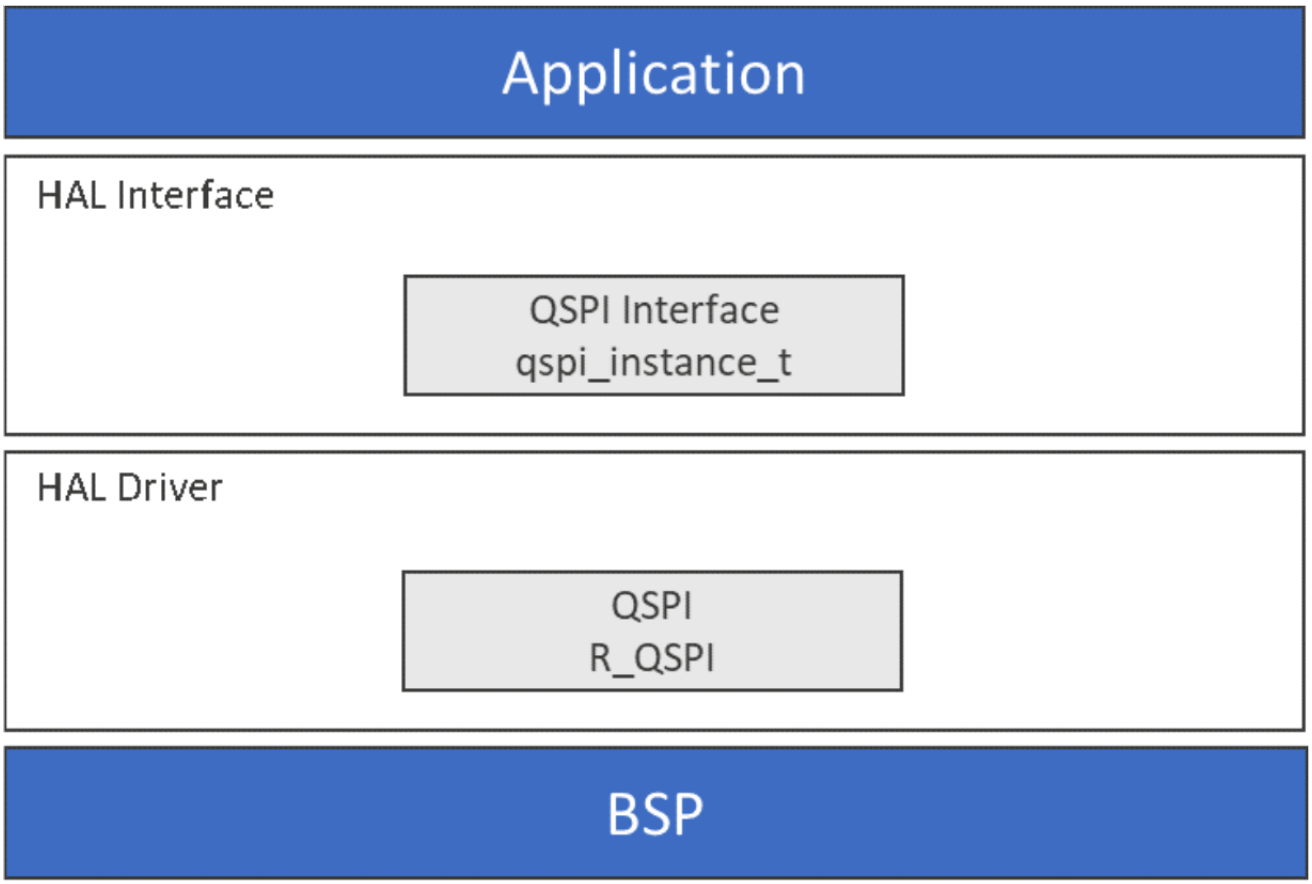


Figure 366: QSPI HAL Module Block Diagram

QSPI Hardware Support Details

The following hardware features are, or are not, supported by SSP for the QSPI.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Extended SPI	Dual SPI	Quad SPI	SPI Mode 0 & 3	Selectable Address (8, 16, 24, 32 bits) via SFMSAC register	Timing adjustment function
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A

S3A1	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S3A3	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S3A6	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S3A7	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S5D3	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S5D5	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S5D9	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒
S7G2	✓	☒	☒	Mode 0	3-byte and 4-byte addresses	☒

MCU Group	Flash read function: Read	Flash read function: Fast Read	Flash read function: Fast Read/Dual Output	Flash read function: Fast Read/Dual I/O	Flash read function: Fast Read/Quad Output	Flash read function: Fast Read/Quad I/O
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	☒	☒	☒	☒	☒	✓
S3A3	☒	☒	☒	☒	☒	✓
S3A6	☒	☒	☒	☒	☒	✓
S3A7	☒	☒	☒	☒	☒	✓
S5D3	☒	☒	☒	☒	☒	✓
S5D5	☒	☒	☒	☒	☒	✓
S5D9	☒	☒	☒	☒	☒	✓
S7G2	☒	☒	☒	☒	☒	✓

MCU Group	Substitutable Instruction Code	Adjustment # of Dummy cycles	Prefetch Function	SPI bus cycle extension Function	Direct communication function	XIP
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	☒	☒	☒	☒	✓	✓
S3A3	☒	☒	☒	☒	✓	✓
S3A6	☒	☒	☒	☒	✓	✓
S3A7	☒	☒	☒	☒	✓	✓
S5D3	☒	☒	☒	☒	✓	✓
S5D5	☒	☒	☒	☒	✓	✓
S5D9	☒	☒	☒	☒	✓	✓
S7G2	☒	☒	☒	☒	✓	✓

4.2.39.2 QSPI HAL Module APIs Overview

The QSPI interface defines API functions for opening, closing, reading, writing, erasing, and device bank selection using the QSPI HAL module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

QSPI HAL Module API Summary

Function Name	Example API Call and Description
open	g_qspi0.p_api->open(g_qspi0.p_ctrl, g_qspi0.p_cfg); Open the QSPI HAL module.
close	g_qspi0.p_api->close(g_qspi0.p_ctrl); Close the QSPI HAL module.
read	g_qspi0.p_api->read(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, readBuffer, BUFFER_LENGTH); Read data from the flash.
pageProgram	g_qspi0.p_api->pageProgram(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, writeBuffer, BUFFER_LENGTH); Program a page of data to the flash.
sectorErase	g_qspi0.p_api->sectorErase(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS); Erase a sector on the flash.

erase	g_qspi0.p_api->erase(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, BYTE_COUNT); Erase a block of memory depending on the input argument "byte_count"
statusGet	g_qspi0.p_api->statusGet(g_qspi0.p_ctrl, &in_progress); Get the write or erase status of the flash.
bankSelect	g_qspi0.p_api->bankSelect(0); Select the bank to access.
infoGet	g_qspi0.p_api->infoGet(g_qspi0.p_ctrl, &qspi_info); Provides information about the underlying QSPI flash, as specified in bsp_qspi.c
versionGet	g_qspi0.p_api->versionGet(& ssp_version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_ASSERTION	p_cfg was NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_UNSUPPORTED	Driver not able to query the following information from the flash manufacturer id, memory capacity and memory type.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.39.3 QSPI HAL Module Operational Overview

The QSPI HAL module is used to initialize the QSPI peripheral so that the Synergy device can communicate (read, write and erase data) with a QSPI serial flash device.

The driver supports three operation modes: page program (write), read and erase.

- The Page program operation programs a single page of data to the flash device. Page size is specific to flash memory and may vary with the vendor to vendor. The typical page size of flash are 128, 256 or 512 bytes. Use qspi_api_t::infoGet API to get the supported page size by the underlying flash device.
- The Read operation will read the data from the flash and store it to the user provided

buffer.
- The Erase operation will erase the block of data from the flash. Use qspi_api_t::infoGet API to get the supported erase size by the underlying flash device.

Note

1. After any Erase/Write operation and before starting the next operation, it is advisable to use the qspi_api_t::statusGet API to poll the status of operation. Not doing so may corrupt user data.
2. For above mentioned operations, the value of function parameter p_device_address should be in the range 0x60000000 – 0x63FFFFFF. Accessing the addresses beyond this range will cause data wrap around. To access the addresses beyond this range, select the valid and appropriate bank using qspi_api_t::bankSelect.

QSPI HAL Module Important Operational Notes and Limitations

QSPI HAL Module Operational Notes

In the case of using a board supported by the SSP and a BSP based project (for example the SK-S7G2 and DK-S7G2) and the board having a QSPI memory device pre-installed, the BSP initializes and places the QSPI peripheral in ROM access mode with XIP (execute in place) enabled. This process enables the memory to be read like standard memory, meaning the QSPI HAL Module is only needed when erasing and programming the QSPI flash device.

Page programming by default uses only one line (D0). To use multiple lines during page program, define the command QSPI_PAGE_PROGRAM_DATA_LINES as one of the following macros in bsp_qspi.h:

- QSPI_COMMAND_PAGE_PROGRAM_QUAD
- QSPI_COMMAND_4BYTE_PAGE_PROGRAM_QUAD
- QSPI_COMMAND_PAGE_PROGRAM_DUAL
- QSPI_COMMAND_4BYTE_PAGE_PROGRAM_DUAL

Also, define QSPI_PAGE_PROGRAM_ADDRESS_ONE_LINE to 1 if the address should be sent on D0 only during dual input page program and quad input page program commands. An example of this can be found in bsp_qspi.h for S7G2-DK starting in SSP version 1.5.0.

The accompanying code to this module guide demonstrates both modes of operation: access via the QSPI HAL Module and ROM access mode with XIP enabled.

The typical QSPI application programs or erases data on the QSPI flash device. When this driver is not open, the QSPI flash device contents get mapped to 0x60000000 and can be read as if ordinary memory.

This driver has been tested on the Micron N25Q256A QSPI flash device.

To configure a QSPI flash memory with 4-byte addressing mode on customized boards, follow the steps below:

1. Make sure QSPI chip supports 4-byte addressing mode by referring the user manual of the flash device.
2. Configure the following in bsp_qspi.c and bsp_qspi.h, referring to the user manual of the flash device.
    - bsp_qspi.c file:

        - Macros:
            - Dummy cycles required
            - Delay timings
            - Page size
        - Global variables:
            - Erase size
            - Erase commands (sector, block, and chip)
    - bsp_qspi.h file:
        - Manufacturer ID
        - Memory Type
        - Memory Capacity
    - Enum definitions:
        - qspi_command (if required)
3. In configuration window select addressing mode as 4-byte.

To use a QSPI flash memory on customized boards, follow the steps below:

1. Configure the following in bsp_qspi.c and bsp_qspi.h, referring to the user manual of the flash device.
    - bsp_qspi.c file:
        - Macros:
            - Dummy cycles required
            - Delay timings
            - Page size
        - Global variables:
            - Erase size
            - Erase commands (sector, block, and chip)
    - bsp_qspi.h file:
        - Manufacturer ID
        - Memory Type
        - Memory Capacity
    - Enum definitions:
        - qspi_command (if required)
2. Calculation of memory capacity in bsp qspi.c file by user (if required).
    - Memory capacity varies for different QSPI flash chips and it should be handled accordingly in bsp_qspi_config_get() function
    - Example: For MT23QL512QSPI FLASH memory capacity calculation should be:

```
uint8_t actual_ memory_capacity = n25_device_characteristics.memory_capacity;
if(actual_ memory_capacity > 0x1F)
{
actual_ memory_capacity = 0x19 + (actual_ memory_capacity - 0x1F);
*p_memory_capacity    = actual_ memory_capacity;
}
else
{
*p_memory_capacity    = n25_device_characteristics.memory_capacity;
}
```

QSPI HAL Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4-Byte supported MCU/board list:

Sl No.	Board	QSPI Flash Part No.	QSPI Memory Size	3-byte hardware support	3-byte software support	4-byte hardware support	4-byte software support	4-byte bsp changes	XIP Mode support	Remarks
1	s7g2_dk (v4.0, v4.1)	MX25L12835F (Macronix)	16MB	YES	YES	NO	NO	YES	YES	4-byte addressing not supported by device
2	s7g2_dk (v3.95 & below)	N25Q256A13E40 (Micron)	32MB	YES	YES	YES	NO	YES	YES	4-byte command is only for part numbers N25Q256A83ESF40x, N25Q256A83E1240x, and N25Q256A83ESFA0F
3	s7g2_sk	W25Q64FV (Winbond)	8MB	YES	YES	NO	NO	YES	YES	4-byte addressing not supported by device
4	s5d3_tbb	MX25L12835F (Macronix)	16MB	YES	YES	NO	NO	YES	YES	4-byte addressing not supported by device

5	s5d5_tbb	N25Q256A13E40 (Micron)	32MB	YES	YES	YES	NO	YES	YES	4-byte command is only for part numbers N25Q256A83ESF40x, N25Q256A83E1240x, and N25Q256A83ESFA0F
6	s5d9_pk	W25Q64FV (Winbond)	8MB	YES	YES	NO	NO	YES	YES	4-byte addressing not supported by device
7	s3a7_dk	N25Q256A83E40 (Micron)	32MB	YES	YES	YES	YES	YES	YES	
8	s3a3_adk	N25Q256A83E40 (Micron)	32MB	YES	YES	YES	YES	YES	YES	
9	s3a1_adk	N25Q256A83E40 (Micron)	32MB	YES	YES	YES	YES	YES	YES	

Note

S1 MCU Series does not support QSPI.

4.2.39.4 Including the QSPI HAL Module in an Application

This section describes how to include the QSPI HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the QSPI Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the QSPI Driver is g_qspi0. This name can be changed in the associated Properties window.)

QSPI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_qspi0 QSPI Driver on r_qspi	Threads	**New> Driver> Storage> QSPI Driver on r_qspi**

When the QSPI Driver on r_qspi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

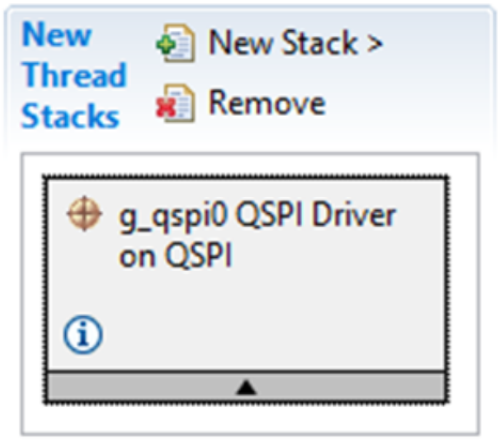


Figure 367: QSPI HAL Module Stack

4.2.39.5 Configuring the QSPI HAL Module

The QSPI HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_qspi0	Module name.
Addressing Mode	3-Byte, 4-Byte Default: 3-Byte	Addressing modes of flash memory For memory > 16 MB, 4-Byte addressing mode should be selected

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

QSPI HAL Module Pin Configuration

The QSPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table indicates the method for selecting pins within the SSP configuration window and the subsequent table lists an example selection sequence for the QSPI pins.

Pin Selection Sequence for the QSPI HAL Module on r_qspi

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select **Peripherals**> **Storage:QSPI**> **QSPI0**

Note

The selection sequence assumes the QSPI0 is the desired hardware target for the driver.

Pin Configuration Settings for the QSPI HAL Module

Property	Value	Description
Pin Group Selection	- Mixed - _A only	Pin group selection.
Operation Mode	- Disabled - Custom - Single or Dual - Quad	Operation mode.
QSPCLK	None, P500	QSPI clock output pin.
QSSL	None, P501	QSPI slave select pin.
QIO0	None, P502	Data 0 input/output.
QIO1	None, P503	Data 1 input/output.

QIO2	None, P504	Data 2 input/output.
QIO3	None, P505	Data 3 input/output.

Note

The example settings come from a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.39.6 Using the QSPI HAL Module in an Application

The typical steps in using the QSPI HAL module in an application are:

1. Initialize the QSPI HAL module using the qspi_api_t::open API call.
2. Read a block of data using the qspi_api_t::read API call.
3. Erase a sector of data using the qspi_api_t::sectorErase API call.
4. Erase n bytes of data using the qspi_api_t::erase API call.
    a. qspi_api_t::infoGet can be used to get the supported erase sizes by the underlying flash.
    b. qspi_api_t::statusGet API can be used to poll the status of ease operation applicable for sectorErase API also.
5. Program a page of data using the qspi_api_t::pageProgram API call.
    a. Use qspi_api_t::infoGet API to get the page size supported by the underlying flash.
    b. qspi_api_t::statusGet API can be used to poll the status of write operation.
6. Close the QSPI HAL module using the qspi_api_t::close API call.

Note

It is advisable to use the qspi_api_t::erase API instead of the qspi_api_t::sectorErase API as a Sector is not a standard size for flash device and it varies with different vendors.

These common steps are illustrated in a typical operational flow diagram in the following figure:

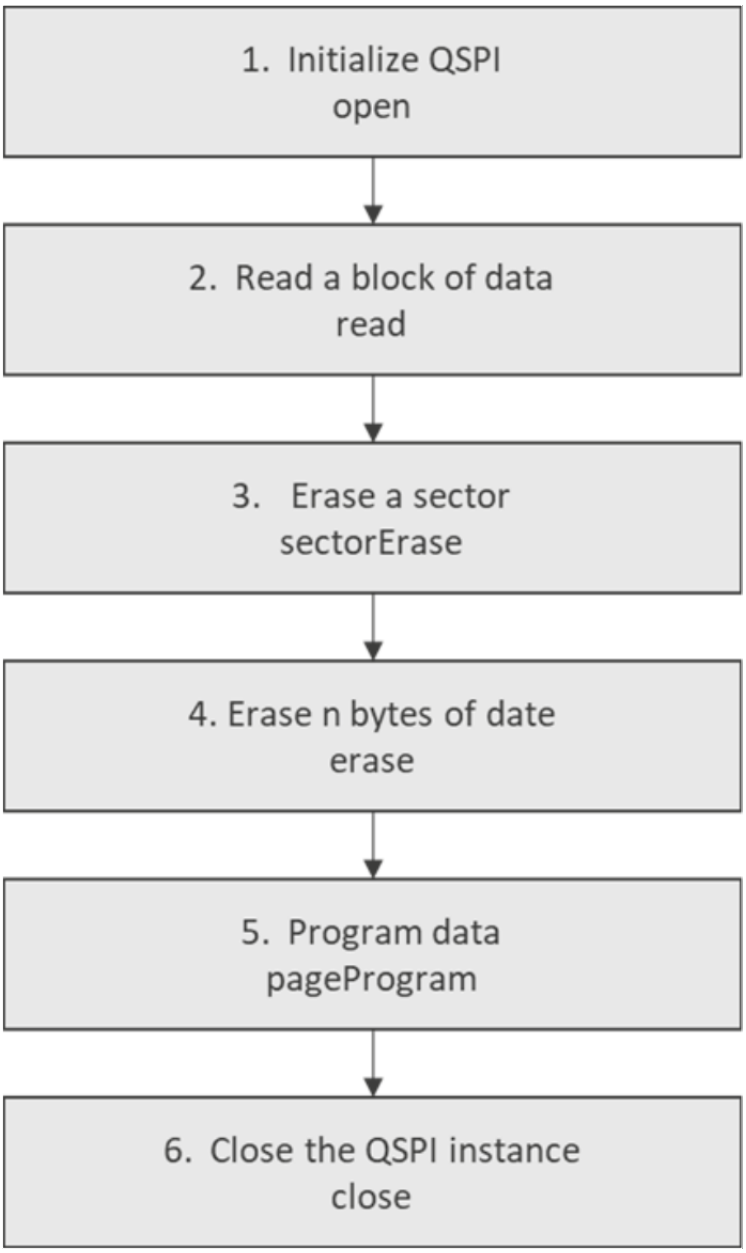


Figure 368: Flow Diagram of a Typical QSPI HAL Module Application

4.2.40 RTC Driver

4.2.40.1 RTC HAL Module Introduction

The Real-Time Clock (RTC) HAL module implements a high-level API for real-time timing applications and uses the real-time clock module on a Synergy MCU. The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to any of the three supported interrupt types: alarm, periodic, or carry.

RTC HAL Module Features

- RTC peripheral configuration.
- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC time counter start and stop.
- RTC alarm, periodic, and carry event notification.
- RTC event type enable and disable.
- RTC event rate configuration.
- RTC clock source set and get.
- RTC sub-clock error adjustment.
- RTC status get.

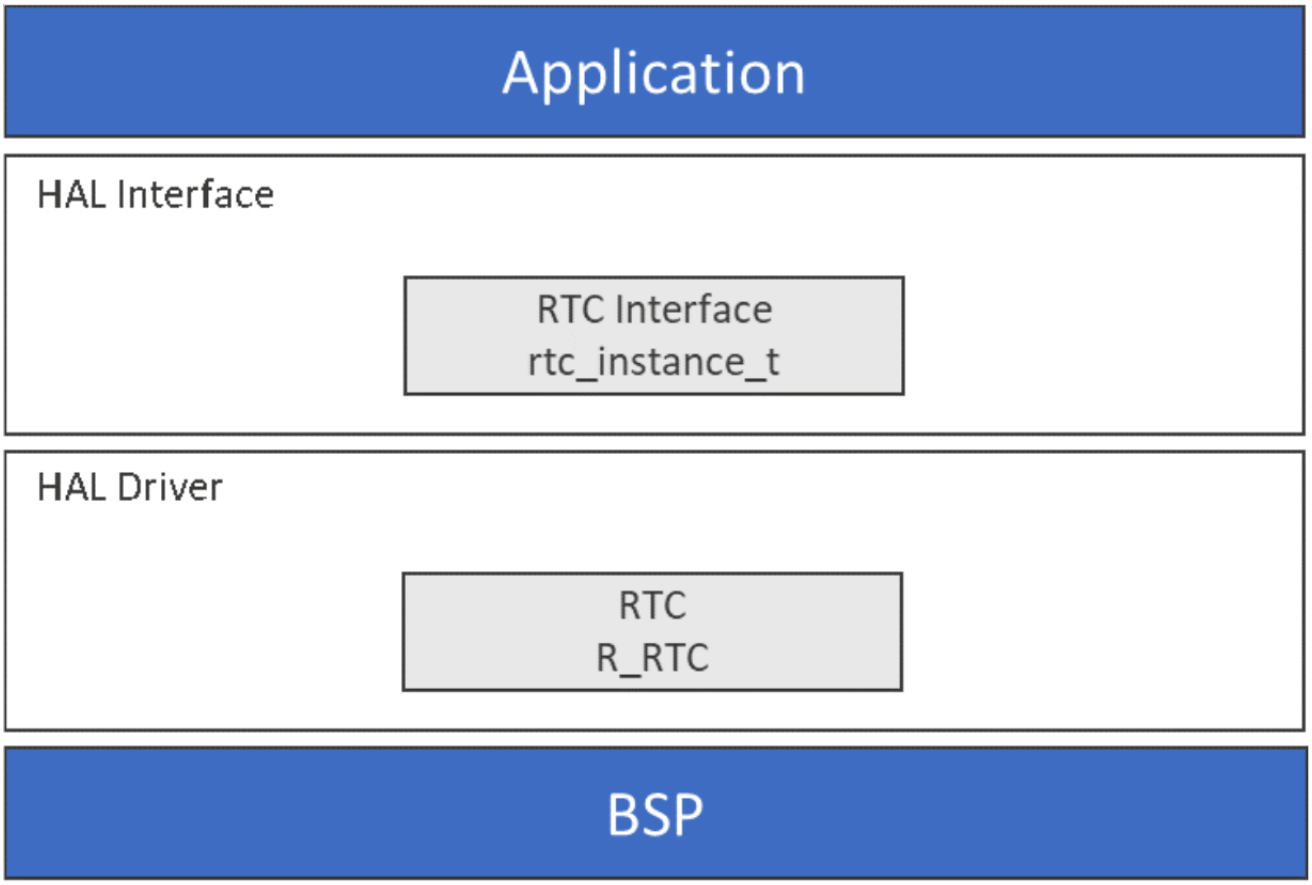


Figure 369: RTC HAL Module Block Diagram

RTC Hardware Support Details

The following hardware features are, or are not, supported by SSP for the RTC.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A	Not supported by MCU

MCU Group	Calendar Mode	Binary Mode	Sub-clock (XCIN) Count Source	LOCO Count Source	12 hours/24 hours	Alarm interrupt (RTC_ALM)
S124	✓	☒	✓	✓	24 Hours	✓
S128	✓	☒	✓	✓	24 Hours	✓
S1JA	✓	☒	✓	✓	24 Hours	✓
S3A1	✓	☒	✓	✓	24 Hours	✓
S3A3	✓	☒	✓	✓	24 Hours	✓
S3A6	✓	☒	✓	✓	24 Hours	✓
S3A7	✓	☒	✓	✓	24 Hours	✓
S5D3	✓	☒	✓	✓	24 Hours	✓
S5D5	✓	☒	✓	✓	24 Hours	✓
S5D9	✓	☒	✓	✓	24 Hours	✓
S7G2	✓	☒	✓	✓	24 Hours	✓

MCU Group	Periodic interrupt (RTC_PRD)	Carry interrupt (RTC_CUP)	Time capture function	Event link function through ELC HAL driver	Start/stop function	Clock error correction function
S124	✓	✓	☒	☒	✓	✓
S128	✓	✓	☒	☒	✓	✓
S1JA	✓	✓	☒	☒	✓	✓
S3A1	✓	✓	☒	☒	✓	✓
S3A3	✓	✓	☒	☒	✓	✓
S3A6	✓	✓	☒	☒	✓	✓
S3A7	✓	✓	☒	☒	✓	✓
S5D3	✓	✓	☒	☒	✓	✓
S5D5	✓	✓	☒	☒	✓	✓
S5D9	✓	✓	☒	☒	✓	✓
S7G2	✓	✓	☒	☒	✓	✓

4.2.40.2 RTC HAL Module APIs Overview

The RTC HAL module defines APIs for opening, closing, setting alarms and starting and stopping RTC

operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

RTC HAL Module API Summary

Function Name	Example API Call and Description
open	g_rtc0.p_api->open(g_rtc0.p_ctrl, g_rtc0.p_cfg); Open the RTC HAL.
close	g_rtc0.p_api->close(g_rtc0.p_ctrl); Close the RTC HAL.
configure	g_rtc0.p_api->configure(g_rtc0.p_ctrl, p_extend);
calendarTimeSet	g_rtc0.p_api->calendarTimeSet(g_rtc0.p_ctrl, &start_time_struct_in, true); Set the calendar time.
calendarTimeGet	g_rtc0.p_api->calendarTimeGet(g_rtc0.p_ctrl, &current_time_struct_out); Get the calendar time.
calendarAlarmSet	g_rtc0.p_api->calendarAlarmSet(g_rtc0.p_ctrl, &in_alarm_time_struct_in, true); Set the calendar alarm time.
calendarAlarmGet	g_rtc0.p_api->calendarAlarmGet(g_rtc0.p_ctrl, &get_alarm_time_struct_out); Get the calendar alarm time.
calendarCounterStart	g_rtc0.p_api->calendarCounterStart(g_rtc0.p_ctrl); Start the calendar counter.
calendarCounterStop	g_rtc0.p_api->calendarCounterStop(g_rtc0.p_ctrl); Stop the calendar counter.
irqEnable	g_rtc0.p_api->irqEnable(g_rtc0.p_ctrl, CALLBACK); Enable the alarm irq.
irqDisable	g_rtc0.p_api->irqDisable(g_rtc0.p_ctrl, CALLBACK); Disable the alarm irq.
periodicIrqRateSet	g_rtc0.p_api->periodicIrqRateSet(g_rtc0.p_ctrl, Rate); Set the periodic irq rate.
infoGet	g_rtc0.p_api->infoGet(g_rtc0.p_ctrl, clk_src); Return the currently configure clock source for the RTC.
errorAdjustmentModeSet	
errorAdjustmentSet	

versionGet	g_rtc0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	API dependent error.
SSP_ERR_INVALID MODE	Invalid mode.
SSP_ERR_INVALID_PTR	Invalid parameter.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.40.3 RTC HAL Module Operational Overview

The RTC HAL module controls the operation of the real-time clock module on a Synergy MCU. The typical RTC application configures the real-time clock controller periodically based on a system configuration driven by the user. Common operations include setting the time, setting an alarm, configuring a periodic interrupt, and starting or stopping operation. An RTC application usually consists of calls to the RTC HAL module and an optional callback from ISR handler.

- The RTC HAL module can use two main clock sources
  - A Low Speed On-Chip Oscillator (LOCO) with lower power, but with less accuracy
  - A sub-clock oscillator with higher power, increased accuracy, and more cost (external crystal required)
- The RTC HAL module supports three different interrupt types
  - An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
  - A periodic interrupt generated every 2, 1, ½, ¼, 1/8, 1/16, 1/32, 1/64 ,1/128 or 1/256 second(s)
  - A carry interrupt when either a carry to the second counter occurs or when a carry to the R64CNT counter occurs during a read access to the 64 Hz counter

A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for any supported interrupt type. When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

Note

The carry interrupt priority must be set to avoid an incorrect time returned from the rtc_api_t::calendarTimeGet API during roll-over.

RTC HAL Module Important Operational Notes and Limitations

RTC HAL Module Operational Notes

The RTC HAL module must be opened before any of the other RTC module APIs can be called. A configuration structure is passed to the open call which specifies the clock source, the name of the user callback from ISR handler, and a user-specified context for the callback. Configuration structures can be either manually defined or generated by the ISDE based on user input during the configuration process.

Functions in the driver can be accessed by either making direct calls to the HAL layer or by using the RTC interface structure. The name of this interface structure is based on the name setting entered in the module's configuration. For example, if the name is g_rtc, then the interface structure is called g_rtc_api.

RTC HAL Module Limitations

- The rtc_api_t::calendarTimeGet API must not be called with interrupts disabled globally, as this API internally uses carry interrupt for its processing. The API may return incorrect time if this is done.
- This module has no support for the following functions:
    - Binary-count mode
    - Binary alarm set and get
    - Binary time get and set.
    - LOCO clock-error correction
    - 1-Hz/64-Hz clock output
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.40.4 Including the RTC HAL Module in an Application

This section describes how to include the RTC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the RTC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the RTC Driver is g_rtc0. This name can be changed in the associated Properties window.)

RTC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_rtc0 RTC HAL on r_rtc	Threads	**New Stack> Driver> Timers> RTC HAL on r_rtc**

When the RTC Driver on r_rtc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and

displays possible choices.

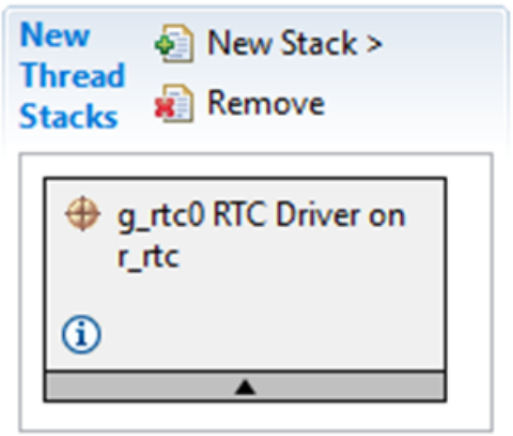


Figure 370: RTC HAL Module Stack

4.2.40.5 Configuring the RTC HAL Module

The RTC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the RTC HAL Module on r_rtc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_rtc0	The name to be used for the RTC module control block instance. This name is also used as the prefix of the other variable instances. See the example code below.
Clock Source	LOCO, Sub-clock Default: LOCO	Clock source for the RTC block.

Configure RTC hardware in open() call	Yes, No Default: Yes	If enabled, the RTC registers and clock source will be initialized in the open() call. If disabled, the user call must call the configure() api to initialize the hardware.
Error Adjustment Value	0	Warning: Deprecated configuration field. Must be 0.
Error Adjustment Type	None	Warning: Deprecated configuration field. Must be 0.
Callback	NULL	The name of the ISR that is called when one of the three interrupts fire. The argument passed into this ISR has an indication of which interrupt caused it to be called. See the example code below.
Alarm Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Alarm interrupt priority selection.
Period Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Period interrupt priority selection.
Carry Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Carry interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

RTC HAL Module Clock Configuration

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator)
  - Lower-power consumption
  - Less accurate
- Sub-clock oscillator
  - Higher-power consumption
  - More accurate
  - More cost (requires a crystal)

The LOCO is the default selection during configuration.

RTC HAL Module Pin Configuration

The RTC does not currently support outputs, so no output pin selections are available.

4.2.40.6 Using the RTC HAL Module in an Application

General Usage

The typical RTC application configures the real-time clock controller periodically based on a system configuration driven by the user. Examples include setting the time, setting an alarm, configuring a periodic interrupt, etc. An RTC application consists of calls to the RTC module and an optional callback.

The RTC module must be opened before any of the other APIs can be called. A configuration structure is passed to the open call which specifies the clock source, the name of the callbacks, and user-specified context for the handler. Configuration structures can be either manually defined or generated by the ISDE based on user input during the configuration process. Functions in the module can be accessed by using the RTC interface structure. The name of this interface structure is based on the name setting entered in the module's configuration.

Avoid Drift Issue After Reset

To avoid drift in RTC time across MCU reset, the application needs to follow the following steps:

Make these changes to module configuration settings:

1. Disable the "Configure Subclock Drive On Reset" option in the CGC stack element of the ISDE configurator.
2. Disable the "Configure RTC hardware in open() call" option in the RTC stack element of the ISDE configurator.

Make these calls in the application code:

1. Call the rtc_api_t::open API as usual.
2. Call the RTC rtc_api_t::configure API only on a cold start. This will initialize the RTC only in a cold start condition.

These two steps can be performed by using the following initialization sequence in the application:

```
    g_rtc.p_api->open(g_rtc.p_ctrl,g_rtc.p_cfg);
    g_rtc.p_api->infoGet(g_rtc.p_ctrl,&info1);
    /* initialize RTC if its status is stopped state i.e. on cold start */
    if(RTC_STATUS_STOPPED == info1.status)
    {
/* if the RTC clock source is sub-clock, stop it so that the sub-clock drive capacity
is set correctly in the configure API */
          g_cgc.p_api->clockStop(CGC_CLOCK_SUBCLOCK);
          g_rtc.p_api->configure(g_rtc.p_ctrl, NULL);
          g_rtc.p_api->calendarTimeSet(g_rtc.p_ctrl,&rt_time,true);
                }
```

An alternative way to determine cold start condition (instead of status) is to obtain the information from the reset status registers (RSTSRx).

Date and Time Validation

The "Parameter Checking" setting needs to be enabled in the ISDE configurator if date and time validation is required for the rtc_api_t::calendarTimeSet and rtc_api_t::calendarAlarmSet APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using rtc_api_t::calendarAlarmSet API, only the fields which have their corresponding match flag set are written to the registers, other register fields are reset to default value.

Sub-Clock Error Adjustment

The rtc_api_t::errorAdjustmentModeSet and rtc_api_t::errorAdjustmentSet APIs can be used to correct the error in the RTC sub-clock source. These APIs can only be used after the RTC is configured and time is set.

The error adjustment is reset every time the RTC is reconfigured or time is set.

There are two common application uses for the RTC HAL module. The first simply uses the RTC to supply the current time as required by the application. A second use of the RTC HAL modules uses the periodic interrupt capability to initiate a process at a regular period. Examples of both uses are provided below.

Note

The configuration property 'Configure RTC hardware in open() call' in the RTC stack of the ISDE configurator controls the behavior of the rtc_api_t::open API. If enabled, the RTC peripheral is configured in the rtc_api_t::open API. If disabled, it is the responsibility of the application to make sure that RTC is configured before usage by using the rtc_api_t::configure API.

The typical steps in using the RTC HAL module in a timing application are:

1. Initialize the RTC using the rtc_api_t::open API
2. Set the time using the rtc_api_t::calendarTimeSet API
3. Set the alarm using the rtc_api_t::calendarAlarmSet API (If required)
4. Start the calendar counter using the rtc_api_t::calendarCounterStart API
5. Get the current time using the rtc_api_t::calendarTimeGet API (as required)

These common steps are illustrated in a typical operational flow diagram in the following figure:

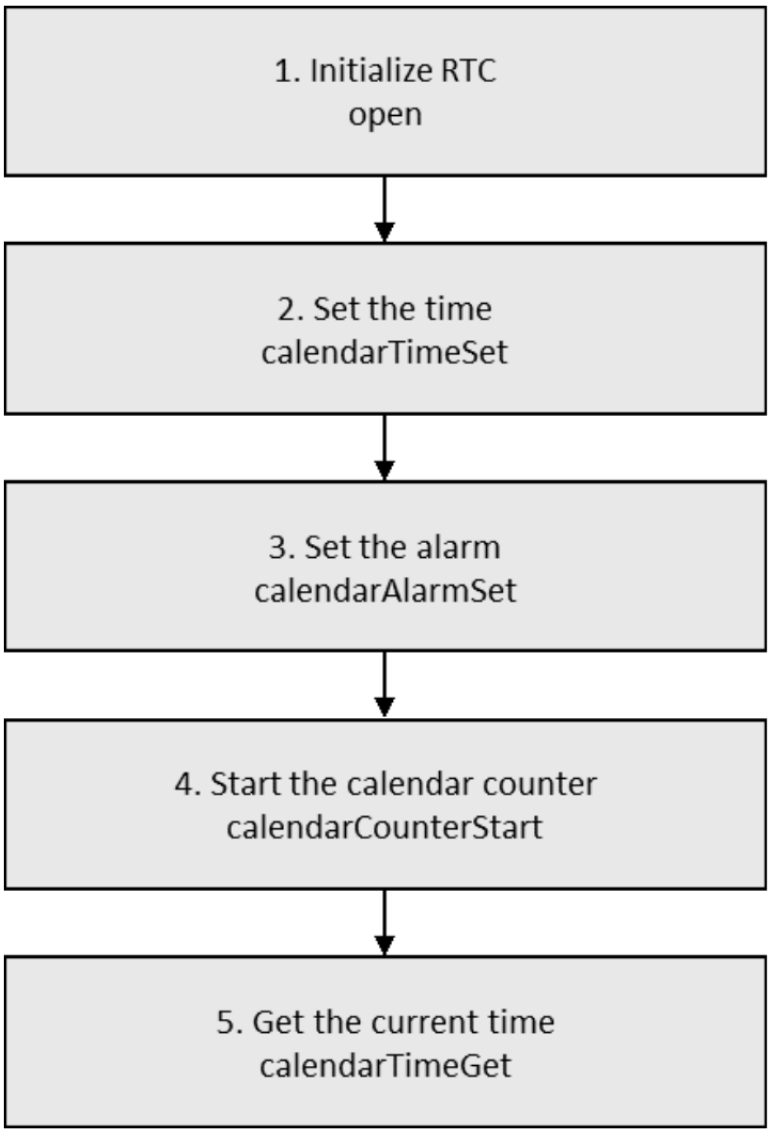


Figure 371: Flow Diagram of a Typical RTC HAL Module Timing Use Application

The typical steps in using the RTC periodic IRQ in an application are:

1. Initialize the RTC using the rtc_api_t::open API.
2. Set periodic IRQ rate using the rtc_api_t::periodicIrqRateSet API.
3. Start calendar counter using the rtc_api_t::calendarCounterStart API.
4. Enable interrupt using the rtc_api_t::irqEnable API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

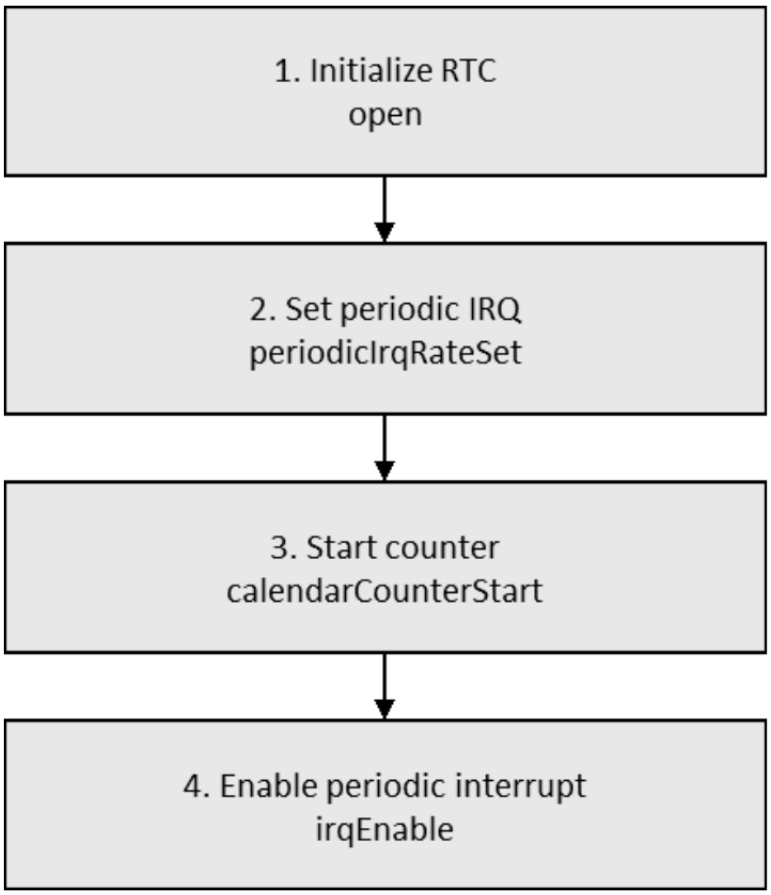


Figure 372: Flow Diagram of a Typical RTC HAL Module Application

4.2.41 SCE Crypto Driver

4.2.41.1 SCE HAL Module Introduction

The Secure Cryptographic Engine (SCE) HAL module provides high-level API functions for random number generation, digest computation (hash), data encryption and decryption, digital signing and verification, key generation (using RSA, AES and ECC algorithms), ECC scalar multiplication and key installation (for RSA, AES and ECC keys). The SCE is a dedicated hardware block and the functionality provided by the SCE varies across the supported MCUs.

SCE HAL Module Features

The SCE HAL module configures the cryptographic module, which allows user to build cryptographic protocols for security with the following cryptographic primitives:

- Random-number generation
- Data encryption and decryption using AES or Triple DES (3DES) or ARC4 algorithms
- Signature generation and verification using the ECC, RSA or DSA algorithms
- Scalar multiplication support for ECDH key agreement operations.
- Message-digest computation using HASH algorithms MD5, SHA1, SHA224, or SHA256

- Key generation - AES wrapped keys, RSA plain text and wrapped keys, ECC plain text and wrapped keys
- Installing the encrypted user key on to the Synergy platform.

The terms "key wrapping" and "key installation" in the context of SSP are defined as follows:

Key Wrapping: The APIs to generate symmetric keys or asymmetric key pairs on the Synergy platform where the private/secret key is a wrapped key (encrypted key).

Key Installation: User generated private /secret keys on a PC (system outside of the Synergy platform) will be installed (no storage) on the Synergy platform and the wrapped private /secret key returned to the user.

Wrapped keys provide the following advantages:

- The wrapped key can only be used on the Synergy platform (MCU) on which it was generated.
- It cannot be moved to another Synergy platform (MCU).
- The original key cannot be recovered from the wrapped key.

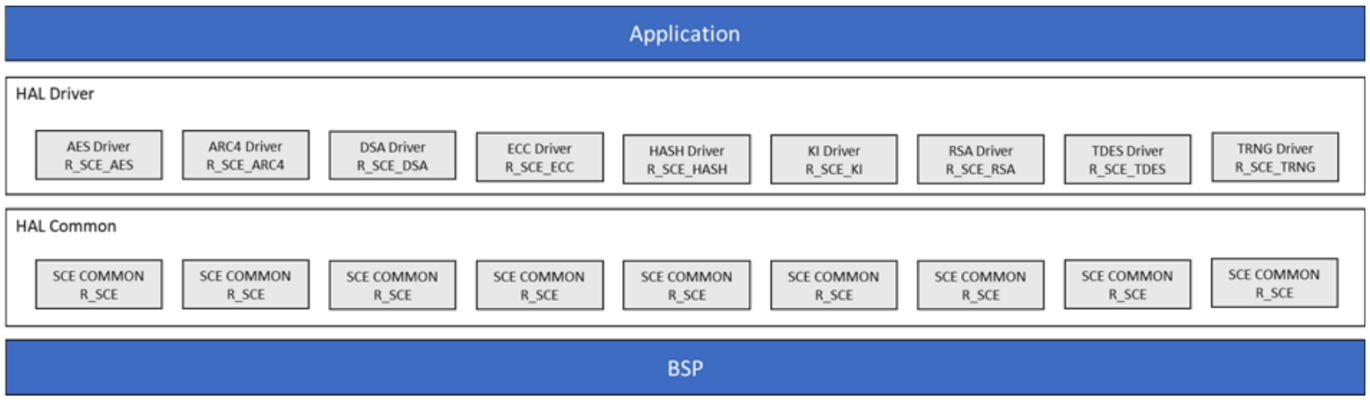


Figure 373: SCE HAL Module Block Diagram

*KI is an abbreviation for Key Installation

SCE Hardware Support Details

Note: The prior figure shows all nine available crypto modules. The SCE COMMON module is repeated for each one since it is included when the module is added to a thread stack. Common modules can be referenced by multiple other module instances across multiple Synergy stacks.

Support of MCU Groups: SCE Driver

Function	S7G2, S5D9, S5D5, S5D3	S3A1, S3A3, S3A7, S3A6	S1JA, S124, **S128**	Notes
TRNG	Generate and read random number	Generate and read random number	Generate and read random number	Generate and read random number

AES	Encryption, decryption, key generation - wrapped keys	Encryption, decryption, key generation - wrapped keys	Encryption, decryption	Symmetric key encryption based on AES standard
AES Key Size	128-bit, 192-bit, 256-bit	128-bit, 256-bit	128-bit, 256-bit	
AES Key Type	Plain text/raw key, wrapped key	Plain text/raw key, wrapped key	Plain text/raw key	
AES Chaining Modes	ECB, CBC, CTR, GCM, XTS††	ECB, CBC, CTR, GCM, XTS	ECB, CBC, CTR	
ARC4	Encryption, decryption	NA	NA	
TDES	Encryption, decryption	NA	NA	
TDES Key Size	192-bit	NA	NA	
TDES Chaining Modes	ECB, CBC, CTR	NA	NA	
RSA	Signature Generation, Signature Verification, Public-key Encryption, Private-key Decryption, Key Generation - plain text and wrapped keys	NA	NA	Supports CRT keys and standard keys for private key operations for both plain-text and wrapped key types
Function	**S7G2, S5D9, S5D5, S5D3**	**S3A1, S3A3, S3A7, S3A6**	**S1JA, S124,** **S128**	**Notes**
RSA Key Size	1024-bit, 2048-bit	NA	NA	
RSA Key Type	Plain text/raw standard format and CRT keys, wrapped standard format and wrapped CRT keys	NA	NA	
Key Installation	AES, ECC, RSA keys	AES keys	NA	

ECC	Key Generation – plain text and wrapped keys, Scalar Multiplication, ECDSA – Signature Generation, ECDSA – Signature Verification	NA	NA	
ECC Key Size (in bits)	192, 224, 256, and 384	NA	NA	
ECC Key Type	Plain text/ raw keys and wrapped keys	NA	NA	
DSA	Signature Generation, Signature Verification	NA	NA	
DSA Key Size	(1024, 160)-bit, (2048, 224)-bit, (2048, 256)-bit	NA	NA	
HASH	MD5, SHA1, SHA224, SHA256	NA	NA	Message digest algorithms

†† XTS is supported for 128-bit and 256-bit keys only.

4.2.41.2 SCE HAL Module APIs Overview

The SCE interface provides a common API for SCE HAL modules. The SCE interface supports multiple operations depending on the chosen module (AES, ARC4, RSA, DSA, HASH, TDES or TRNG).

The AES interface defines APIs for opening, closing, generating wrapped keys, encrypting and decrypting data using the AES algorithm. It uses a 128-bit, 192-bit or 256-bit key and ECB, CBC, CTR, GCM or XTS chaining-mode options. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. For return status values, refer to the SCE API reference section of the SSP User's Manual.

SCE Common Instance API Summary

Function Name	Example API Call and Description
open	g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg); SCE Common module open function. Must be called before performing any other crypto operations.
close	g_sce.p_api->close(g_sce.p_ctrl); Close the SCE Common module.

interfaceGet	g_sce.p_api->interfaceGet(g_sce_aes.p_ctrl, p_interface_info, p_interface); Get the interface structure for the interface info provided.
statusGet	g_sce.p_api->statusGet (g_sce.p_ctrl, p_status); Get status of SCE initialization.
versionGet	g_sce.p_api->versionGet(&version); Gets the module code and API version and stores it in provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

AES HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_aes.p_api->open(g_sce_aes.p_ctrl, g_sce_aes.p_cfg); AES module open function. Must be called before performing any encrypt/decrypt operations.
createKey	g_sce_aes.p_api->createKey(g_sce_aes.p_ctrl, num_words, p_key); Generate an AES key for encrypt/decrypt operations.
encrypt	g_sce_aes.p_api->encrypt(g_sce_aes.p_ctrl, p_key, p_vi, num_words, p_source, p_dest); AES encryption using the chaining mode and padding mode specified in the open() function call.
addAdditionalAuthenticationData	g_sce_aes.p_api->addAdditionalAuthenticationData (g_sce_aes.p_ctrl, p_key, p_vi, num_words, p_source); Add additional authentication data (called before starting an encryption or decryption operation).
encryptFinal	g_sce_aes.p_api->encryptFinal(g_sce_aes.p_ctrl, p_key, p_iv, input_num_words, p_source, output_num_words, p_dest); AES final encryption using the chaining mode and padding mode specified in the open() function call.
decrypt	g_sce_aes.p_api->decrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_words, p_source, p_dest); AES decryption.

setGcmTag	g_sce_aes.p_api->setGcmTag(g_sce_aes.p_ctrl,num_words, p_source); Set authentication tag data.
getGcmTag	g_sce_aes.p_api->getGcmTag(g_sce_aes.p_ctrl,num_words, p_dest); Get authentication tag data.
zeroPaddingEncrypt	g_sce_aes.p_api->zeroPaddingEncrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_bytes, p_source, p_dest) AES zero padding encryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage - 1. To provide any Add Authentication Data (AAD): set p_dest = NULL 2. Encryption: set p_source to input data and p_dest will return encrypted data 3. Get/Compute Tag: set p_source = NULL
zeroPaddingDecrypt	g_sce_aes.p_api->zeroPaddingDecrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_words, p_source, p_dest); AES zero padding decryption< using the chaining mode and padding mode specified. Implementation for GCM mode only API usage - 1. Set expected tag value using the setGcmTag() function 2. To provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL 3. Decryption: set p_source to input encrypted data, decrypted data will be returned in p_dest 4. To verify the tag, invoke this API using p_source = NULL and p_dest = NULL, the return value indicates authentication tag verification status.
versionGet	g_sce_aes.p_api->versionGet(&version); Gets the module code and API version and stores it in provided version pointer.
close	g_sce_aes.p_api->close(g_sce_aes.p_ctrl); Close the AES module.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The ARC4 interface defines APIs for opening, closing, setting a key and processing data. A complete list of the available APIs, an example API call and a short description of each can be found in the following table:

ARC4 HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_arc4.p_api->open(g_sce_arc4.p_ctrl, g_sce_trng.p_cfg); Open the ARC4 module.
keySet	g_sce_arc4.p_api->keySet(g_sce_arc4.p_ctrl, &rngbuf, nbytes); Set the key to be used by the ARC4 module.
arc4Process	g_sce_arc4.p_api->arc4Process(g_sce_arc4.p_ctrl, nbytes, &source, &destination); Encrypt or decrypt data using the ARC4 module.
close	g_sce_arc4.p_api->close(g_sce_arc4.p_ctrl); Close the ARC4 module.
versionGet	g_sce_arc4.p_api->versionGet (&version); Retrieve the version using the provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The DSA interface defines APIs for opening, closing, digital-signing and verification. Available options include a 1024-bit public key and a 160-bit private key, a 2048-bit public key and a 224-bit private key or a 2048-bit public key and a 256-bit private key. A complete list of the available APIs, an example API call and a short description of each can be found in the following table:

DSA HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_dsa.p_api->open(g_sce_dsa.p_ctrl, g_sce_dsa.p_cfg); DSA module open function. Must be called before performing any sign/verify operations.
verify	g_sce_dsa.p_api->verify(p_key, p_domain, num_words, p_signature, p_paddedHash); DSA signature verification using given DSA public key. This function is deprecated. The function hashVerify should be used instead.
hashVerify	g_sce_dsa.p_api->hashVerify(g_sce_dsa.p_ctrl, p_key, p_domain, num_words, p_signature, p_paddedHash); DSA signature verification using given DSA public key.

sign	g_sce_dsa.p_api->sign(p_key, p_domain, num_words, p_padded_hash, p_dest); DSA Signature generation using DSA private key. This function is deprecated. The function hashSign should be used instead.
hashSign	g_sce_dsa.p_api->hashSign(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); DSA Signature generation using DSA private key.
close	g_sce_dsa.p_api->close(g_sce_dsa.p_ctrl); Close the DSA module.
versionGet	g_sce_dsa.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

ECC HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_ecc.p_api->open(g_sce_ecc.p_ctrl, g_sce_ecc.p_cfg); Open the ECC driver. This API must be called before performing any ECC operations.
close	g_sce_ecc.p_api->close(g_sce_ecc.p_ctrl); Close the ECC module.
scalarMultiplication	g_sce_ecc.p_api->scalarMultiplication(g_sce_ecc.p_ctrl, p_domain, p_k, p_p, p_r); This API calculates R=kP.
keyCreate	g_sce_ecc.p_api->keyCreate(g_sce_ecc.p_trl, p_domain, p_generator_point, p_key_private, p_key_public); This API generates key pair for ECC.
sign	g_sce_ecc.p_api->sign(g_sce_ecc.p_ctrl, p_domain, p_generator_point, p_key_private, msg_digest, signature_r, signature_s); This API generates signature of ECDSA.
verify	g_sce_ecc.p_api->verify(g_sce_ecc.p_ctrl, p_domian, p_generator_point, p_key_public, msg_digest, signature_r, signature_s); This is a procedure for signature verification of ECDSA.

versionGet	g_sce_ecc.p_api->versionGet(&version); Gets version and stores it in provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The HASH interface defines APIs for calculating hash values for a given data-set. Available options include SHA1 and SHA256 algorithms. A complete list of the available APIs, an example API call and a short description of each can be found in the following table:

HASH HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_hash.p_api->open(g_sce_hash.p_ctrl, g_sce_hash.p_cfg); HASH module open function. Must be called before performing any sign/verify operations.
updateHash	g_sce_hash.p_api->updateHash(p_source, num_words, p_dest); Update hash for the num_words words from source buffer p_source. This function is deprecated. The function hashUpdate should be used instead.
hashUpdate	g_sce_hash.p_api->hashUpdate(g_sce_hash.p_ctrl, p_source, num_words, p_dest); Update hash for the num_words words from source buffer p_source.
close	g_sce_hash.p_api->close(g_sce_hash.p_ctrl); HASH module close function.
versionGet	g_sce_hash.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Key Installation HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_key_installation.p_api->open(g_sce_key_installation.p_ctrl, p_cfg); Open the Crypto Key Installation framework for subsequent call/Key installation.

close	g_sce_key_installation.p_api->close(g_sce_key_installation.p_ctrl); Close the Crypto Key Installation framework.
keyInstall	g_sce_key_installation.p_api->keyInstall (g_sce_key_installation.p_ctrl, p_user_key_input, p_user_key_rsa_modulus, p_install_key_input, p_key_data_out); Install a key version 2. This function takes the RSA modulus of the user's RSA private key as one of the input parameters to return the RSA wrapped key in a format that is compatible with other Crypto APIs. For all other key types the functionality remains the same as the earlier keyInstall API.
versionGet	g_sce_key_installation.p_api->versionGet(&version); Get version of the Crypto Key Installation framework and stores it in the provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The RSA interface defines APIs for opening, closing, encrypting and decrypting data using an RSA algorithm as well as digitally signing and verifying the algorithm. The RSA interface employs a 1024-bit or 2048-bit key. A complete list of the available APIs, an example API call and a short description of each can be found in the following table:

RSA HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_rsa.p_api->open(g_sce_rsa.p_ctrl, g_sce_rsa.p_cfg); RSA module open function. Must be called before performing any encrypt/decrypt or sign/verify operations.
encrypt	g_sce_rsa.p_api->encrypt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Encrypt source data from p_source using an RSA public key from p_key and write the results to destination buffer p_dest.
decrypt	g_sce_rsa.p_api->decrypt (g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest.

decryptCrt	g_sce_rsa.p_api->decryptCrt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. RSA private key data is specified in CRT format.
verify	g_sce_rsa.p_api->verify(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_signature, p_padded_hash); Verify signature given in buffer p_signature using the RSA public key p_key for the given padded message hash from buffer p_padded_hash.
sign	g_sce_rsa.p_api->sign(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. Write the results to the buffer p_dest.
signCrt	g_sce_rsa.p_api->signCrt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. RSA private key p_key is assumed to be in CRT format. Write the results to the buffer p_dest.
close	g_sce_rsa.p_api->close(g_sce_rsa.p_ctrl); Close the RSA module.
keyCreate	g_sce_rsa.p_api->keyCreate(g_sce_rsa.p_ctrl, p_private_key, p_public_key); Generates an RSA key pair.
versionGet	g_sce_rsa.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The TDES interface defines APIs for encrypting and decrypting data according to the TDES standard. A complete list of the available APIs, an example API call and a short description of each can be found in the following table:

TDES HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_tdes.p_api->open(g_sce_tdes.p_ctrl, g_sce_tdes.p_cfg); Open the TDES module.

encrypt encrypt	g_sce_tdes.p_api->encrypt(g_sce_tdes.p_ctrl, &key, &iv, nwords, &source, &destination); Encrypt the data.
decrypt	g_sce_tdes.p_api->decrypt(g_sce_tdes.p_ctrl, &key, &iv, nwords, &source, &destination); Decrypt the data.
close	g_sce_tdes.p_api->close(g_sce_tdes.p_ctrl); Close the TDES module.
versionGet	g_sce_tdes.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

The TRNG interface defines APIs for computing the random-number generator. A complete list of the available APIs, an example API call and a short description of each can be found in the following table.

TRNG HAL Module API Summary

Function Name	Example API Call and Description
open	g_sce_trng.p_api->open(g_sce_trng.p_ctrl, g_sce_trng.p_cfg); Open the TRNG driver for reading random data from the hardware TRNG module.
read	g_sce_trng.p_api->read(g_sce_trng.p_ctrl, p_rngbuf, nbytes); Generate nbytes of random number bytes and store them in p_rngbuf buffer.
close	g_sce_trng.p_api->close(g_sce_trng.p_ctrl); Close the TRNG interface driver.
versionGet	g_sce_trng.p_api->versionGet(&version); Gets version and stores it in provided version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

4.2.41.3 SCE HAL Module Operational Overview

Different cryptographic functions are available for different target MCUs; the following table shows the functionality that is available for each individual MCU-series:

Function	S7G2, S5D9, S5D5, S5D3	S3A1, S3A3, S3A7, S3A6	S124, S128, S1JA	Notes
TRNG	Generate and read random number	Generate and read random number	Generate and read random number	Generate and read random number.
AES	Encryption, decryption, Key Generation - wrapped keys	Encryption, decryption, Key Generation - wrapped keys	Encryption, decryption	Symmetric Key Encryption based on AES standard.
AES Key Size	128-bit, 192-bit, 256-bit	128-bit, 256-bit	128-bit, 256-bit	
AES Key Type	Plain text/raw key, Wrapped key	Plain text/raw key, Wrapped key	Plain text/raw key	
AES Chaining Modes	ECB, CBC, CTR, GCM, XTS Note: XTS is supported for 128-bit and 256-bit keys only	ECB, CBC, CTR, GCM, XTS	ECB, CBC, CTR	
ARC4	Encryption, decryption	NA	NA	
TDES	Encryption, decryption	NA	NA	
TDES Key Size	192-bit	NA	NA	
TDES Chaining Modes	ECB, CBC, CTR	NA	NA	
RSA	Signature Generation, Signature Verification, Public-key Encryption, Private-key Decryption, Key Generation - plain text and wrapped keys	NA	NA	Supports CRT keys and standard keys for private key operations for both plain-text and wrapped key types.
RSA Key Size	1024-bit, 2048-bit	NA	NA	
RSA Key Type	Plain text/raw standard format and CRT keys, wrapped standard format and wrapped CRT keys	NA	NA	

DSA	Signature Generation, Signature Verification	NA	NA	
DSA Key Size	(1024, 160)-bit, (2048, 224)-bit, (2048, 256)-bit	NA	NA	
HASH	MD5, SHA1, SHA224, SHA256	NA	NA	Message digest algorithms.
ECC	Key Generation – plain text and wrapped keys, Scalar Multiplication, ECDSA- Signature Generation, ECDSA -Signature Verification,	NA	NA	
ECC Key Size	192-bit, 224-bit, 256-bit and 384-bit	NA	NA	
ECC Key Type	Plain Text/Raw Key, Wrapped Key			
Key Installation	AES, ECC, RSA keys	AES keys	NA	

Configuration Settings for the R_SCE Module

The endianness of the SCE is set to big endian by default. It can be set to little endian mode.

Please refer to the operational notes on endianness configuration parameter usage.

Configuration Settings for the TRNG Module

Random number-generation can be configured for the maximum number of attempts it makes to the underlying hardware to generate a unique 16-byte random number that differs from the previously-generated random number. On reaching the maximum number of attempts, the read API will return an error code to the caller, otherwise a success code is returned and the generated random number will be transferred to the caller-supplied data buffer.

Configuration Settings for the AES Module

The AES module can be configured for a user-specified key-length, key type (plain text or wrapped key) and chaining modes.

Configuration Settings for the RSA Module

The RSA module can be configured for a user-specified key length and key type: plain text or wrapped keys.

Configuration Settings for the DSA Module

The DSA module can be configured for a user-specified key length.

Configuration Settings for the HASH Module

The HASH module can be configured for a user specified HASH algorithm (depending on the target MCU.)

Configuration Settings for the TDES Module

The TDES module can be configured for a user-specified chaining mode.

Configuration Settings for the Key Installation Module

The Key Installation module can be configured to install the user's encrypted key.

Configuration Settings for the ECC Module

The ECC module can be configured for a user-specified key length and key type.

SCE HAL Module Important Operational Notes and Limitations

SCE HAL Module Operational Notes

- Synergy S7 and S5 devices have the SCE7 and therefore support AES, TRNG, RSA, HASH, DSA, ECC and Key Installation.
- Synergy S3 devices have the SCE5 and therefore support AES, TRNG, & GHASH. GHASH is supported as part of the AES GCM mode. Key Installation is supported as part of AES GCM, ECB, CTR, XTS, CBC chaining modes. Synergy S3 devices do not support MD5, SHA1/SHA256 HASH functionality.
- Synergy S1 devices only support AES and TRNG.
- If an unsupported module is added to the project, then a compiler warning will be generated indicating this fact.
- All modules support the versionGet API which can be called even before a module is opened.
- R_SCE module crypto_api_t::interfaceGet API is provided for use by the Framework layer SF CRYPTO modules.

The InterfaceGet API is used to request a crypto HAL interface. The example below shows usage of this API:

```
crypto_instance_t         * p_crypto;  /* R_SCE  instance */
void * p_interface = NULL;    /* Declare a pointer to hold the output interface
structure object */
crypto_interface_get_param_t     param;
param.hash_type = CRYPTO_TYPE_HASH_256; /* Requesting SHA 256 interface*/
/* It is mandatory for the address of the p_interface pointer be passed to the API */
p_crypto->p_api->interfaceGet(&param, &p_interface);
```

- All crypto APIs may return SSP_ERR_ASSERTION on null pointer input or invalid input parameters. All APIs return error codes documented in sf_crypto_err_t or ssp_err_t which are within the width of the type uint32_t.
- Crypto hardware engine does not support reentrancy. When the crypto hardware engine is busy performing a task, any new request will receive a status error code SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT.

Endianness configuration parameter usage:

- The default mode is big endian where the input and output parameters (example: keys, payload and IV) are required to be in uint32_t data type.
- The little endian mode allows the user to have uint8_t/byte array for input and output parameters (example: keys, payload and IV) and they should be cast to (uint32_t *).
- The endianness configuration is set at the initialization of the SCE module and remains in effect until the module is closed. Hence all data should be formatted accordingly.

Example:

- Select the Big endian mode when the data is in uint32_t and big endian format:

uint32_t test_data[5] = {0x84983E44, 0x1C3BD26E, 0xBAAE4AA1, 0xF95129E5, 0xE54670F1};

- Select the Little endian mode when the same data is in byte array format

```
uint8_t  test_data_byte_array[20] =
{0x84, 0x98, 0x3E, 0x44, 0x1C, 0x3B, 0xD2, 0x6E, 0xBA, 0xAE, 0x4A, 0xA1, 0xF9, 0x51,
0x29, 0xE5, 0xE5, 0x46, 0x70, 0xF1};
```

AES Keys:

AES wrapped key sizes are as follows:

```
/* Return Wrapped AES secret key size in bytes for a 128-bit AES Key */
#define AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES  (36U)
/* Return Wrapped AES secret key size in bytes for a 192-bit AES Key */
#define AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return Wrapped AES secret key size in bytes for a 256-bit AES Key */
#define AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return Wrapped AES-XTS secret key size in bytes for a 128-bit AES XTS Mode Key */
#define AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52U)
/* Return AES-XTS secret key size in bytes for a 256-bit AES XTS Mode Key */
#define AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES  (84U)
```

The format of RSA keys generated by the rsa_api_t::keyCreate API is as follows:

Note: The endianness is the same as that set during SCE initialization.

RSA Key Format:

RSA Public Key Format:

WORD 0 : Public key exponent

WORD 1: Start of RSA modulus

(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private Key in Plain Text Standard Format:

WORD 0: Private key exponent (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

Followed by RSA modulus. (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private Key in Plain Text CRT Format:

The components are ordered in the following order with exponent2 at byte 0:

exponent2 // the second factor's CRT exponent, a positive integer

prime2 // the second factor, a positive integer

exponent1 // the first factor's CRT exponent, a positive integer

prime1 // the first factor, a positive integer

coefficient // the (first) CRT coefficient, a positive integer

The format of RSA wrapped keys generated by the rsa_api_t::keyCreate API is as follows:

RSA Public key is always in plain text.

Byte 0 to Byte 3: Public key exponent

Byte 4 : Start of RSA modulus

(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private key in standard format

Byte 0: Private key exponent is wrapped.(Length is 148 bytes for RSA 1024-bit and 276 bytes for RSA 2048-bit keys)

- Followed by RSA modulus in plain text. (Length is 128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

SCE HAL Module Limitations

- The AES encrypt() and decrypt() functions do not support data padding. These functions

operate on data lengths that are multiples of 16 bytes. (Data padding needs to be handled by the user application.) AES GCM mode may require support for authentication data that may not be a multiple of 16 bytes. To support this, zeroPaddingEncrypt() and zeroPaddingDecrypt() function APIs are provided only for the AES GCM mode.
    - For AES GCM, when using uint32_t arrays in big endian mode, if the number of bytes of data is not a multiple of 4 (WORD length) it should be zero-padded.
- AES encryption/ decryption API with XTS chaining mode supports data input lengths which are multiples of AES block size. Arbitrary input data lengths will work only when the input data is zero-padded in the user application. (Data padding needs to be handled by the user).
- The TDES encrypt() and decrypt() functions do not support data padding. These functions operate on data lengths that are multiples of 8 bytes. (Data padding needs to be handled by the user application.)
- Disable the unused interfaces in the r_sce module configuration in ISDE properties which will reduce the code memory. This will help in reducing code/text memory size in low memory devices like S5D3 MCUs.

HASH Module - MD5

- MD5 requires byte swapping of the final message digest output. Intermediate updates (partial updates) are not required to be byte swapped.
- MD5 also requires the length field within the formatted final block to be in big endian format before calling the hash_api_t::hashUpdate API.

Refer to the most recent SSP release notes for the most up-to-date limitations on this module.

4.2.41.4 Including the SCE HAL Module in an Application

This section describes how to include the SCE HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Crypto Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table.

SCE HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sce_aes_0 AES Driver on r_sce_aes	Threads	**New Stack> Driver> Crypto> AES Driver on r_sce_aes**
g_sce_arc4_0 ARC4 Driver on r_sce_arc4	Threads	**New Stack> Driver> Crypto> ARC4 Driver on r_sce_arc4**
g_sce_dsa_0 DSA Driver on r_sce_dsa	Threads	**New Stack> Driver> Crypto> DSA Driver on r_sce_dsa**

g_sce_ecc_0 ECC Driver on r_sce_ecc	Threads	**New Stack> Driver> Crypto> ECC Driver on r_sce_ecc**
g_sce_hash_0 HASH Driver on r_sce_hash	Threads	**New Stack> Driver> Crypto> HASH Driver on r_sce_hash**
g_sce_key_initialization_0 Key Initialization Driver on r_sce_key_initialization	Threads	**New Stack> Driver> Crypto> Key Initialization Driver on r_sce_key_initialization**
g_sce_rsa_0 RSA Driver on r_sce_rsa	Threads	**New Stack> Driver> Crypto> RSA Driver on r_sce_rsa**
g_sce_tdes TDES Driver on r_sce_tdes	Threads	**New Stack> Driver> Crypto> TDES Driver on r_sce_tdes**
g_sce_trng TRNG Driver on r_sce_trng	Threads	**New Stack> Driver> Crypto> TRNG Driver on r_sce_trng**

When a Crypto HAL module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

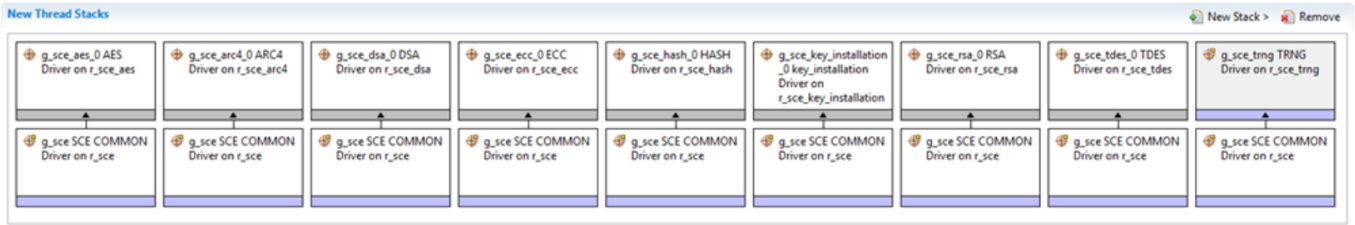


Figure 374: SCE HAL Module Stack

4.2.41.5 Configuring the SCE HAL Module

The SCE HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties

tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the AES HAL Module on r_sce_aes

ISDE Property	Value	Description
Name	g_sce_aes_0	Module name.
Key Length	128, 192, 256 Default: 128	Key length used for encryption/decryption operations by this instance of the driver.
Chaining Mode	ECB, CBC, CTR, GCM, XTS Default: CBC	Block cipher chaining mode used for encryption/decryption operations by this instance of the driver.
Key Format	Plain Text Key, Wrapped Key (Not available for S1 MCU series) Default: Plain Text Key	Key format selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the ARC4 HAL Module on r_sce_arc4

ISDE Property	Value	Description
Name (for S7G2, S5D9, S5D5 devices only)	g_sce_arc40	Module name.
Key Length in number of bytes	0	Key length selection.
Key Name, this symbol must be defined as uint8_t array type data in user code	g_arc4_0_key	Key name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DSA HAL Module on r_sce_dsa

ISDE Property	Value	Description
Name	g_sce_dsa_0	Module name.

Key Length	(1024, 160), (2048, 224), (2048, 256) Default: (2048, 256)	Key length used for signing/verification operations by this instance of the driver.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the ECC HAL Module on r_sce_ecc

ISDE Property	Value	Description
Name (for S7G2, S5D9, S5D5 devices only)	g_sce_ecc0	Module name.
Key Length	192, 224, 256, 384 Default: 256	Key length used for encryption/decryption operations by this instance of the driver.
Key Format	Plain Text Key, Wrapped Key Default: Plain Text Key	Key format selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the HASH Driver on r_sce_hash

ISDE Property	Value	Description
Name (for S7G2, S5D9, S5D5 devices only)	g_sce_hash_0	Module name.
Algorithm	SHA1, MD5, SHA224 SHA256 Default: SHA256	Algorithm used for computing the message digest/hash on the message data.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the RSA HAL Module on r_sce_rsa

ISDE Property	Value	Description
Name	g_sce_rsa_0	Module name.
Key Length	1024, 2048 Default: 2048	Key length used for signing/verification/encryption/decryption operations by this instance of the driver.
Key Format	Plain Text Key, Wrapped Key Default: Plain Text Key	Key format selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the TDES HAL Module on r_sce_tdes

ISDE Property	Value	Description
Name	g_sce_tdes_0	Module name.
Chaining Mode	EBC, CBC, CTR Default: CBC	Chaining mode selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the TRNG HAL Module on r_sce_trng

ISDE Property	Value	Description
Name	g_sce_trng	Module name.
Max. Attempts	2	Sets the maximum number of attempts when a newly generated random number differs from the previously generated random number.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Key Installation on r_sce_key_installation

ISDE Property	Value	Description
Name (Not Supported for S1 Series MCUs)	g_sce_key_installation_0	Module name.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

4.2.41.6 Using the SCE HAL Module in an Application

The SCE Driver on r_sce HAL module makes several APIs available for various cryptographic functions. The steps to use each function are illustrated as follows, but a flow diagram is not provided due to the large number of steps.

The steps in using the SCE Driver on r_sce HAL module in a typical application are:

1. To use the SCE module:
    - Initialize the SCE and the SCE HAL module (R_SCE) using the crypto_api_t::open API through the SCE common driver. This initializes the module as defined in the

associated configuration parameters.

- *Note*

  Configure the endianness (little-endian or big-endian) for the input /output data for all the HAL APIs. The big-endian mode is configured by default. See the above operational notes for details on endianness configuration.

  - All interfaces supported on the selected MCU are available at run-time by calling the crypto_api_t::interfaceGet API. This is available for use by the Crypto framework.
  - If the crypto_api_t::interfaceGet API function is used by the HAL module, disable the unused interfaces. This will exclude them from the build to save code/text space.
  - If the application is using the Crypto frameworks, disable only the interfaces which will not be used at run-time. Make sure to keep all the used interfaces enabled.
  - If the crypto_api_t::interfaceGet API is not used directly by the HAL application project and if none of the Crypto Framework modules are being used, disable all the interfaces. This will reduce the code/text space significantly.
  - Note that disabling the interfaces will only exclude them from being requested through the crypto_api_t::interfaceGet API function. These interfaces can still be used by the HAL project by including the appropriate module in ISDE.
  - Call the crypto_api_t::open API function before using the crypto_api_t::interfaceGet API.

- *Note*

  The open function cannot be called again until the module is closed.

2. To use the AES functions:
   - Initialize the selected AES module with the aes_api_t::open API. This initializes the module as defined in the associated configuration parameters.

     - *Note*

       AES available key sizes are 128-bit, 192-bit or 256-bit. Chaining modes supported are ECB, CBC, CTR, GCM and XTS.

       - Encrypt data with the aes_api_t::encrypt API.
   - Decrypt data with the aes_api_t::decrypt API.
   - Generate keys using the aes_api_t::createKey API.

     - *Note*

       The aes_api_t::createKey API in AES module creates AES wrapped keys. AES plain text keys can be generated with the services of the TRNG module.

   - Close the interface instance using the aes_api_t::close API.

     - *Note*

       Subtle difference exist in GCM operations as follows.

   - IV provided for AES GCM operations must be a 96-bit IV formatted to 128-bits.

Example:

- 96-bit IV formatted to 128-bits: e0e00f19fed7ba0136a797f300000001
- 96-bit IV: e0e00f19fed7ba0136a797f3

AES GCM operations:

The IV will be updated after each operation and that value should be used for each subsequent operation.

AES GCM encryption:

1. AAD (Additional Authenticated Data) is optional. If it is to be used, it has to be provided prior to encrypting/decrypting any data by setting p_dest = NULL
2. Encryption: set p_source to input data and p_dest will return encrypted data
3. Get/Compute Tag: set p_source = NULL.

AES GCM decryption:

1. Set expected tag value using the setGcmTag() function
2. Provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL
3. **Decryption**: set p_source to input encrypted data, decrypted data will be returned in p_dest
4. To verify the tag, invoke this API using p_source = NULL and p_dest = NULL, the return value indicates authentication tag verification status.The decrypted data is to be used only if the tag is verified successfully. zeroPaddingEncrypt/zeroPaddingDecrypt APIs can be used for GCM operations when the data is not a multiple of the block size.
5. To use the TDES functions:
    - Initialize the selected TDES module with the tdes_api_t::open API. This initializes the module as defined in the associated configuration parameters.

    - *Note*

      The TDES chaining mode can be specified as ECB, CBC or CTR.
        - *Encrypt data using the tdes_api_t::encrypt API.*
        - *Decrypt data using the tdes_api_t::decrypt API*
        - *Close the interface instance with tdes_api_t::close API.*
6. To use the ARC4 functions:
    - Initialize the selected ARC4 module with the arc4_api_t::open API. This initializes the module as defined in the associated configuration parameters.

        - *Note*

          The ARC4 key can be specified by length (anywhere from 64-bits, 2048-bits) and location.
    - Set the key with the arc4_api_t::keySet API.
    - Encrypt or decrypt data using the arc4_api_t::arc4Process API.
    - Close the module using the arc4_api_t::close API.
7. To use the RSA functions:
    - Initialize the selected module with the rsa_api_t::open API. This initializes the module as defined in the associated configuration parameters.

        - *Note*

          For rsa_api_t::encrypt API, rsa_api_t::decrypt API, rsa_api_t::decryptCrt API, rsa_api_t::sign and rsa_api_t::signCrt API, the size of the data buffer is indicated in num_words. It must be 32 words /128 bytes/1024-bits for 1024-bit keys and 64 words /256 bytes/2048-bits for the 2048-bit keys.

        - *Note*

          Supported key-formats are Standard Key and CRT Key.

        - *Note*

          Supported key-types are plain-text and wrapped private keys.

- *Note*

  Supported key-lengths are 1024-bits and 2048-bits.

  - Encrypt data with the RSA public Key using the rsa_api_t::encrypt API .
  - Decrypt data with the RSA private Key using the rsa_api_t::decrypt API.
  - Decrypt data with the RSA private Key, in the CRT format, using the rsa_api_t::decryptCrt API.
  - Generate the signature for a given padded hash using the RSA private Key, in the standard format, using the rsa_api_t::sign API.
  - Generate the signature for a given padded hash using the RSA private Key, in the CRT format, using the rsa_api_t::signCrt API.
  - Verify the signature for a given padded hash using the RSA public Key, in the standard format, using the rsa_api_t::verify API.
  - Generate keys using the rsa_api_t::keyCreate API.

    - *Note*

      The rsa_api_t::keyCreate API in the RSA module creates RSA plain-text keys or wrapped keys based on the input parameters to the API.

  - Close the interface instance with the rsa_api_t::close API.

8. To use the DSA functions:
   - Initialize the selected DSA module with the dsa_api_t::open API. This initializes the module as defined in the associated configuration parameters.

     - *Note*

       Supported key-lengths are (1024,160)-bits, (2048,224)-bits and (2048,256)-bits

   - Generate the signature with the DSA private key using the dsa_api_t::hashSign API.
   - Verify the signature with the DSA public key using the dsa_api_t::hashVerify API.
   - Close the module using the dsa_api_t::close API.
9. To use the HASH algorithms:
   - Initialize the selected HASH module with the hash_api_t::open API. This initializes the module as defined in the associated configuration parameters.

     - *Note*

       MD5, SHA1 and SHA256 hash methods are supported.

   - Compute the message digest using the hash_api_t::hashUpdate API.
   - Close the module with hash_api_t::close API.
10. To use the True Random Number Generator functions:
    - Initialize the TRNG module using the trng_api_t::open API.
    - Generate a random number using the trng_api_t::read API.
    - Close the interface instance using the trng_api_t::close API.
11. To use the Key Installation API:
    - Initialize the Key Installation module using the sf_crypto_key_installation_api_t::open API. This initializes the module as defined in the associated configuration parameters.

      - *Note*

        Specify Output key structure with pointer to buffer and buffer length.

      - *Note*

        Specify Key installation key structures for the user's encrypted key and Renesas provided key index (key size, key format, pointer to buffer and buffer length).

    - Install the key using the sf_crypto_key_installation_api_t::keyInstall API.
    - Close the module using the sf_crypto_key_installation_api_t::close API.
12. To use the ECC functions:
    - Initialize the selected ECC module using the ecc_api_t::open API. This initializes the

module as defined in the associated configuration parameters.

- *Note*

  To generate the domain parameters for NIST curves, use OpenSSL command to generate curves as shown below.
  - *ECC P-384: openssl ecparam -name secp384r1 -param_enc explicit -text | more*
  - *ECC P-256: openssl ecparam -name secp256r1 -param_enc explicit -text | more*
  - *ECC P-224: openssl ecparam -name secp224r1 -param_enc explicit -text | more*
  - *ECC P-192: openssl ecparam -name secp192r1 -param_enc explicit -text | more*

- *Note*

  Supported key sizes are 192 bits, 224 bits, 256 bits, and 384 bits.

- *Note*

  For the ecc_api_t::scalarMultiplication API, the ecc_api_t::keyCreate, the ecc_api_t::sign and the ecc_api_t::verify API:
  - *The size of the data buffer is indicated in the data_length field of r_crypto_data_handle_t. The actual buffer must be pointed to by p_data field of the r_crypto_data_handle_t.*

  - Perform ECC Scalar Multiplication using the ecc_api_t::scalarMultiplication API.
  - Generate the signature for a given padded hash using the ECC private Key, in the standard format, using the ecc_api_t::sign API.
  - Verify the signature for a given padded hash using the ECC public Key, in the standard format, using the ecc_api_t::verify API.
  - Generate the ECC keys using the ecc_api_t::keyCreate API.
  - Close the module with the ecc_api_t::close API.

13. Close the SCE and the SCE HAL moduleusing the crypto_api_t::close API.

4.2.42 SDADC Driver

4.2.42.1 SDADC HAL Module Introduction

The SDADC HAL module provides a high level API for analog-to-digital conversions and supports the SDADC24 24-bit analog-to-digital converter peripheral available on the Synergy microcontroller hardware. A user-defined callback can be created to process the data each time a new sample is available.

SDADC HAL Module Features

- 24-bit sigma delta A/D Converter
- Single scan or continuous scan operation mode
- Single-ended or differential input
- Gain of up to 32 on differential inputs
- Oversampling ratio configurable on differential inputs

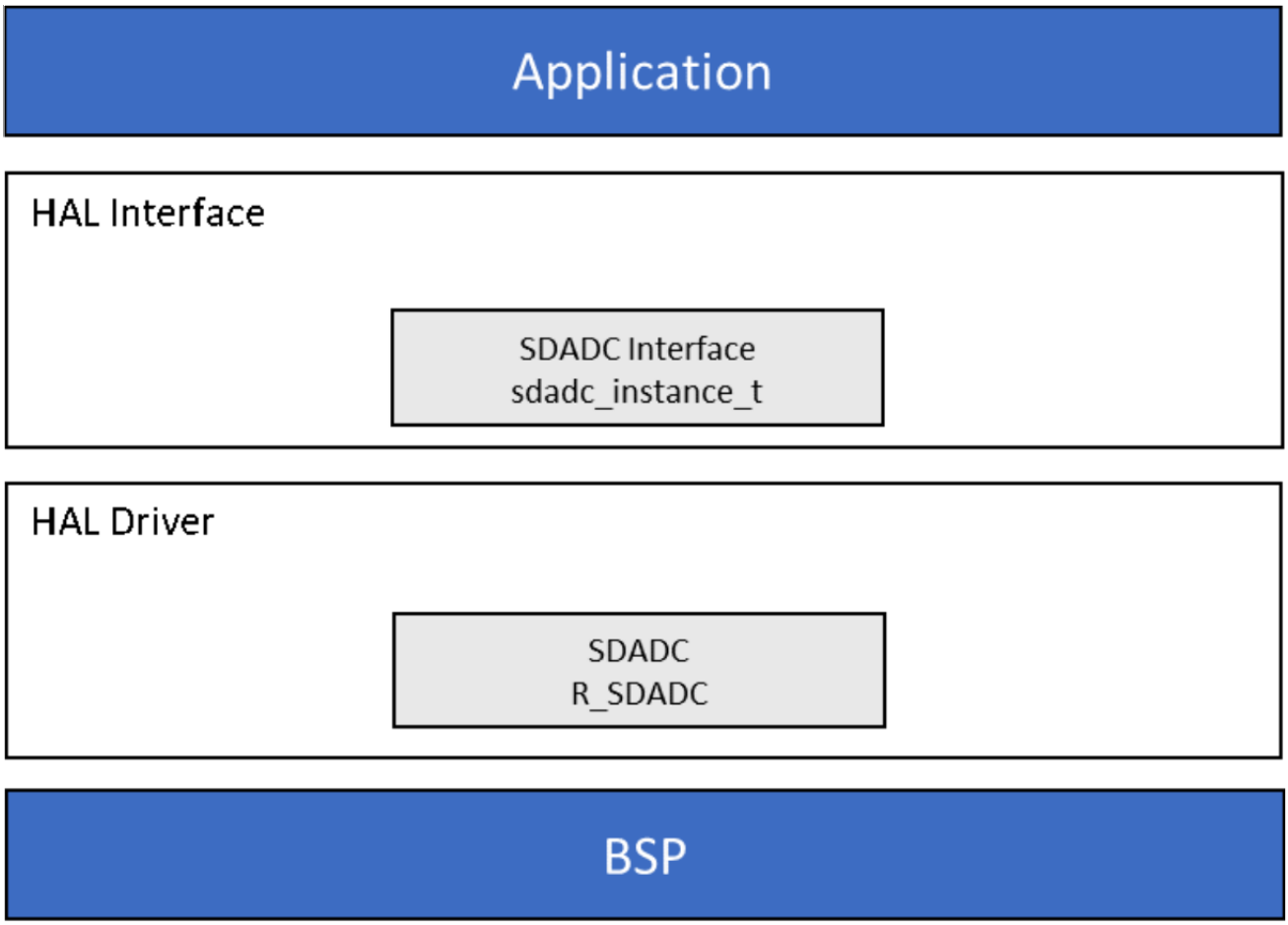


Figure 375: SDADC HAL Module Block Diagram

SDADC Hardware Support Details

The following hardware features are, or are not, supported by SSP for SDADC:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Support for all Analog Channels	24-Bit	Single-scan Mode	Continuous-scan mode
S124	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A
S1JA	✓	✓	✓	✓
S3A1	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A

S3A6	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A

MCU Group	Single-ended input	Differential input	Programmable Gain Amplifier	Configurable oversampling ratio
S124	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A
S1JA	✓	✓	✓	✓
S3A1	N/A	N/A	N/A	N/A
S3A3	N/A	N/A	N/A	N/A
S3A6	N/A	N/A	N/A	N/A
S3A7	N/A	N/A	N/A	N/A
S5D3	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A

4.2.42.2 SDADC HAL Module APIs Overview

The SDADC HAL module defines API functions to open, configure scans, start scans, stop scans, read the conversion results the ADC scans and close the ADC unit. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SDADC HAL Module API Summary

Function Name	Example API Call and Description
open	g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg); Initialize ADC unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.
scanCfg	g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.p_channel_cfg); Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call.

scanStart	g_adc.p_api->scanStart(g_adc.p_ctrl); Start the scan (in case of a software trigger), or enable the hardware trigger.
scanStop	g_adc.p_api->scanStop(g_adc.p_ctrl); Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.
scanStatusGet	g_adc.p_api->scanStatusGet(g_adc.p_ctrl); Check scan status.
read	g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result.
read32	g_adc.p_api->read32(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result into a 32-bit word.
sampleStateCountSet	g_adc.p_api-> sampleStateCountSet(g_adc.p_ctrl,&adc_sample); Set the sample state count for the specified channel.
calibrate	g_adc.p_api-> calibrate(g_adc.p_ctrl, reg_id, offset); Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input.
offsetSet	g_adc.p_api-> offsetSet(g_adc.p_ctrl, p_extend); Set offset for input PGA configured for differential input.
close	g_adc.p_api->close(g_adc.p_ctrl); Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.
infoGet	g_adc.p_api->infoGet(g_adc.p_ctrl, &adc_info); Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read for the DTC/DMAC to read the conversion results of all configured channels.
versionGet	g_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description

SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform R_ADC_Close first.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_CALIBRATION_FAILED	Calibration failed.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.42.3 SDADC HAL Module Operational Overview

The SDADC HAL module controls the SDADC peripheral on a Synergy microcontroller. It directly controls the SDADC hardware without using any RTOS elements and provides convenient APIs to simplify development.

In this document, the term 'scan' refers to the AUTOSCAN feature of the SDADC, which works as follows:

1. Conversions are performed on enabled channels in ascending order of channel number. All conversions required for a single channel are completed before the sequencer moves to the next channel.
2. Conversions are performed at the rate (in Hz) of the SDADC oversampling clock frequency / oversampling ratio (configured per channel). The SSP uses the normal mode SDADC oversampling clock frequency.
3. If averaging is enabled for the channel, the number of conversions to average are performed before each conversion end interrupt occurs.
4. If the number of conversions for the channel is more than 1, performs the number of conversions requested. If averaging is enabled for the channel, each averaged result counts as a single conversion.
5. Continues to the next enabled channel only after completing all conversions requested.
6. After all enabled channels are scanned, a scan end interrupt occurs.

The driver supports single-scan and continuous scan operation modes.

- Single-scan mode performs one scan per trigger (hardware trigger or software start using adc_api_t::scanStart).
- In continuous scan mode, the scan is restarted after each scan completes. A single trigger is required to start continuous operation of the SDADC.

Interrupts and Callbacks

When a conversion is complete and a callback is provided by the user, the SDADC HAL module calls the callback (adc_cfg_t::p_callback) with the argument adc_callback_args_t, indicating the unit and the event adc_cb_event_t.

The SDADC driver supports the following callback events:

- ADC_EVENT_CONVERSION_COMPLETE to notify the application that new conversion data is available.
- ADC_EVENT_SCAN_COMPLETE to notify the application when a scan is complete.
- ADC_EVENT_CALIBRATION_COMPLETE to notify the application that the calibration process is complete.

SDADC HAL Module Important Operational Notes and Limitations

SDADC HAL Module Operational Notes

Triggering a Data Transfer with the SDADC

To trigger a transfer of data when the SDADC scan completes, configure the data transfer with transfer_cfg_t::activation_source set to ELC_EVENT_SDADCn_SCAN_END. The ELC events are listed under elc_event_t. To retrieve the SDADC specific information to use with the Transfer Interface API, use the adc_api_t::infoGet function call.

Triggering ELC Events with the SDADC

The SDADC unit can trigger the start of other peripherals listed in elc_peripheral_t. Refer to the "ELC Interface" in the SSP User's Manual for more information.

SDADC HAL Module Limitations

This module only works for selected Synergy MCUs. Consult the release notes for your current SSP release to see which MCUs are supported by this module. Additionally, the MCU Hardware Manual shows which peripherals are available.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.42.4 Including the SDADC HAL Module in an Application

This section describes how to include the SDADC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SDADC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SDADC Driver is g_sdadc0. This name can be changed in the associated Properties window.)

SDADC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sdadc0 SDADC Driver on r_adc	Threads	**New Stack> Driver> Analog> SDADC Driver on r_sdadc**

When the SDADC Driver on r_sdadc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are

individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

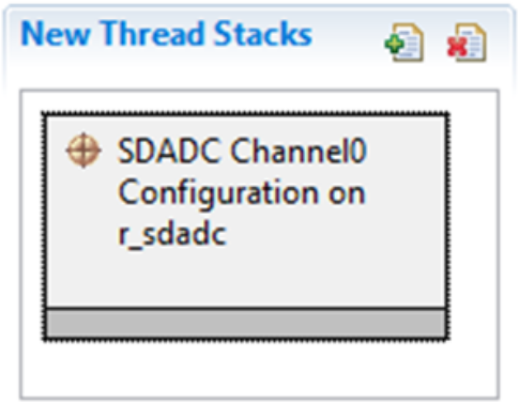


Figure 376: SDADC HAL Module Stack

4.2.42.5 Configuring the SDADC HAL Module

The SDADC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SDADC HAL Module on r_sdadc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_adc0	Module name.

Mode	Single Scan, Continuous Scan Default: Continuous Scan	In single scan mode, all channels are converted once per start trigger, and conversion stops after all enabled channels are scanned. In continuous scan mode, conversion starts after a start trigger, then continues until stopped in software.
Resolution	16 Bit, 24 Bit Default: 24 Bit	Select 24-bit or 16-bit resolution.
Alignment	Right, Left Default: Right	Select left or right alignment.
Trigger	ELC Hardware Event, Software Default: Software	Select conversion start trigger. Conversion can be started in software, or conversion can be started when a hardware event occurs if the hardware event is linked to the SDADC peripheral using the ELC API.
Vref Source	Internal, External Default: Internal	Vref can be sourced internally and output on the SBIAS pin, or Vref can be input from VREFI.
Vref Voltage	0.8V, 1.0V, 1.2V, 1.4V, 1.6V, 1.8V, 2.0V, 2.2V, 2.4V Default: 1.0V	Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%.
Internal Calibration During Open()	Enabled, Disabled Default: Enabled	Calibration is required for all channels configured for differential input. Internal calibration is performed automatically during open for these channels unless it is disabled here.
Callback	NULL	Enter the name of the callback function to be called when conversion completes or a scan ends.
Conversion End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Select the interrupt priority for the conversion end interrupt. [Required]
Scan End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the scan end interrupt. [Required]

Calibration End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the calibration end interrupt. [Required]

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SDADC HAL Module Clock Configuration

The SDADC HAL module uses the SDADCCLK as its clock source and it must be set to 4 MHz.

SDADC HAL Module Pin Configuration

To use the SDADC HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the SDADC HAL Module on r_sdadc

Resource	ISDE Tab	Pin selection Sequence
SDADC	Pins	Select **Peripherals**> **Analog:SDADC**> **SDADCn,** where n is the channel number.

4.2.42.6 Using the SDADC HAL Module in an Application

The typical steps in using the SDADC HAL module in an application are:

1. Initialize the SDADC using adc_api_t::open. Calibration is performed on all channels configured for differential input during this function unless calibration during open is disabled in the configuration.
2. If calibration during open is disabled in the configuration, calibrate each channel configured for differential input using adc_api_t::calibrate. Wait for a calibration complete interrupt after calibrating each channel. See R_SDADC_Calibrate for details regarding the calibration procedure.
3. Configure active channels using adc_api_t::scanCfg. (Optional)
4. Start a conversion using the desired trigger with adc_api_t::scanStart.
    a. If a hardware trigger is used, this call enables the ADC unit to be triggered by the hardware trigger. If a software trigger is used, then this call starts the ADC scan.
5. The callback will be called after each conversion, and when the entire scan is complete.
6. Read the results of the conversion using adc_api_t::read.
7. Stop the ADC scan by calling adc_api_t::scanStop. (Optional)
    a. This prevents the ADC from being triggered by an external trigger or a hardware trigger; it also forces a stop of a software-triggered scan if one is ongoing.
8. Operate on the received data as needed by the application.
9. Close the module and power down the peripheral using the adc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

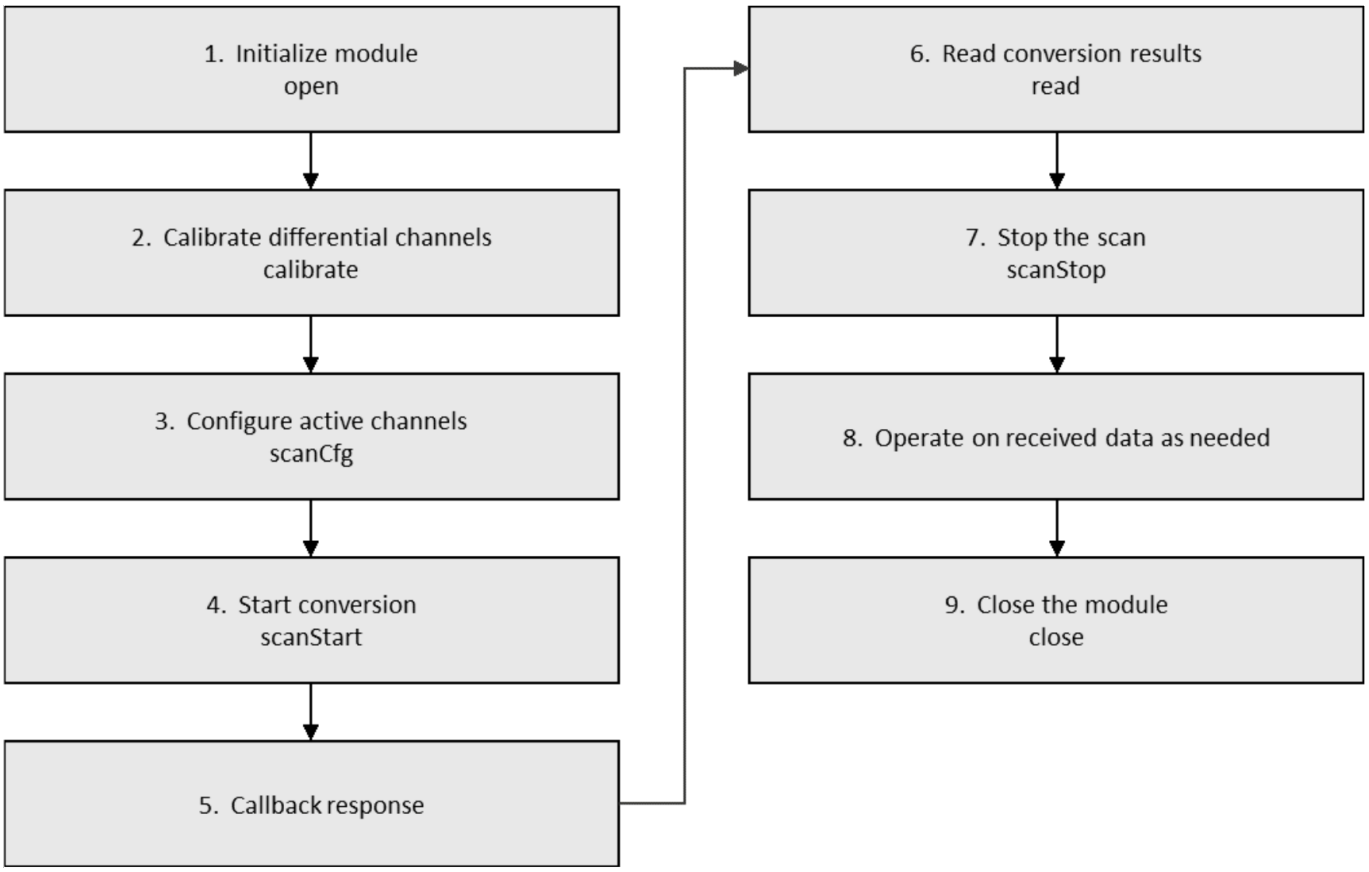


Figure 377: Flow Diagram of a Typical SDADC HAL Module Application

4.2.43 SD/MMC Driver and SDIO Driver

4.2.43.1 SDMMC HAL Module Introduction

The SDMMC (SD/MMC and SDIO) HAL module is used to read/write and control SDMMC media devices and SDIO cards. The SDMMC module can be used as a standalone SD card, eMMC or media driver, or it can be used with FileX and other compatible file systems. The SDMMC HAL module uses the SDMMC peripheral on the Synergy MCU.

SDMMC HAL Module Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity) and eMMC (embedded Multi Media Card)
    - Supports reading, writing and erasing SD and eMMC memory devices
    - Supports 1, 4 or 8-bit data busses (8-bit bus is supported for eMMC only)
    - Supports detection of hardware write protection (SD cards only)
    - Automatically selects between backwards compatible mode and high speed SRD mode (eMMC)
- Supports SDIO
    - Supports SDIO single register access (CMD52)
    - Supports SDIO multiple register access (CMD53)

- Supports SDIO interrupts
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device

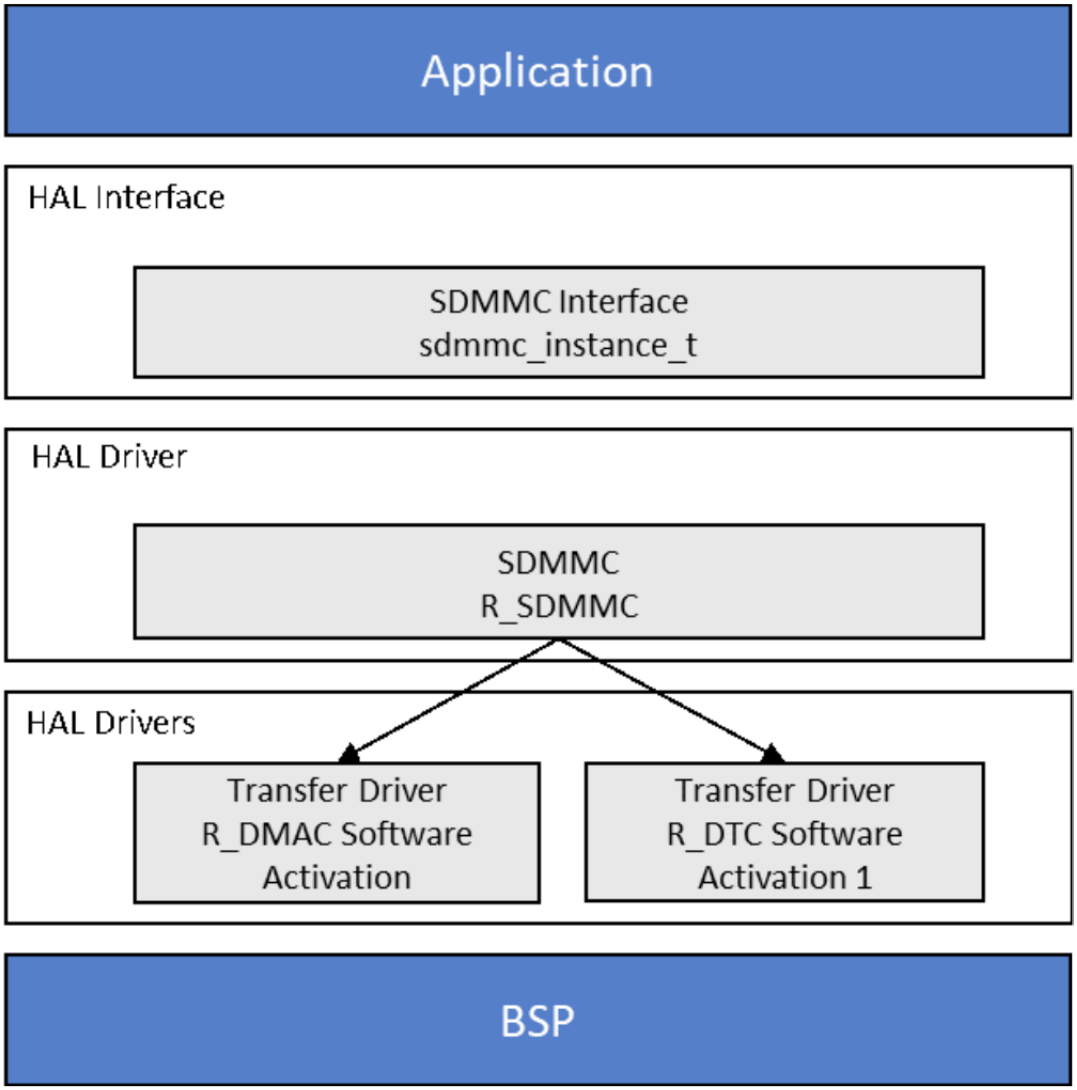


Figure 378: SDMMC HAL Module Block Diagram

SDMMC Hardware Support Details

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	SD 1 bit	SD 4 bit	SDHC	SDXC	MMC 1 bit	MMC 4 bit

S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	✓	✓	✓	✓	☒	☒
S3A3	✓	✓	✓	✓	☒	☒
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	✓	✓	✓	✓	☒	☒
S5D3	✓	✓	✓	✓	☒	☒
S5D5	✓	✓	✓	✓	☒	☒
S5D9	✓	✓	✓	✓	☒	☒
S7G2	✓	✓	✓	✓	☒	✓

MCU Group	MMC 8 bit	MMC Backward Compatibility Mode	Card Detect: Insertion	Card Detect: Removal	Write Protect	MMC High Speed SDR Mode
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	☒	☒	☒	☒	✓	☒
S3A3	☒	☒	☒	☒	✓	☒
S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	☒	☒	☒	☒	✓	☒
S5D3	☒	☒	See note	See note	✓	☒
S5D5	☒	☒	See note	See note	✓	☒
S5D9	☒	☒	See note	See note	✓	☒
S7G2	✓	☒	See note	See note	✓	✓

Note: Available on some of the MCUs (for the group indicated in the table) based on the pin map.

MCU Group	DMA Read	DMA Write	Single Block Read	Single Block Write	IO RW Direct	IO RW Extended
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓

S3A6	N/A	N/A	N/A	N/A	N/A	N/A
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

4.2.43.2 SDMMC HAL Module APIs Overview

The SDMMC HAL module defines API functions including opening, reading, writing and closing. A complete list of the available API functions, an example API function call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SDMMC HAL Module API Summary

Function Name	Example API Call and Description
open	g_sdmmc.p_api->open(g_sdmmc.p_ctrl, g_sdmmc.p_cfg); Open device channel for read/write and control. Initialize driver and hardware on first call.
read	g_sdmmc.p_api->read(g_sdmmc.p_ctrl, &destination, start, count); Read data from SD/MMC channel.
write	g_sdmmc.p_api->write(g_sdmmc.p_ctrl, &source, start, count); Write data to SDMMC channel.
control	g_sdmmc.p_api->control(g_sdmmc.p_ctrl, command, &data); Send control commands to the SD/MMC port and receive the status of the SD/MMC port.
close	g_sdmmc.p_api->close(g_sdmmc.p_ctrl); Close open SDMMC channel. Turns off hardware if last channel open.
versionGet	g_sdmmc.p_api->versionGet(&version); Get version of Block Media SD/MMC driver.
readIo	g_sdmmc.p_api->readIo(g_sdmmc.p_ctrl, &data, function, address); Read I/O data from an SDMMC channel.
writeIo	g_sdmmc.p_api->writeIo(g_sdmmc.p_ctrl, &data, function, address, read_after_write); Write I/O data to SDMMC channel.

readIoExt	g_sdmmc.p_api->readIoExt(g_sdmmc.p_ctrl, &destination, function, address, count, transfer_mode, address_mode); Read I/O data, extended, from an SDMMC channel.
writeIoExt	g_sdmmc.p_api->writeIoExt(g_sdmmc.p_ctrl, &source, function, address, count, transfer_mode, address_mode); Write I/O data, extended, to SDMMC channel.
IoIntEnable	g_sdmmc.p_api->IoIntEnable(g_sdmmc.p_ctrl, enable); Enables SDIO interrupt for SDMMC channel.
infoGet	g_sdmmc.p_api->infoGet(g_sdmmc.p_ctrl, &info); Get SDMMC channel info.
erase	g_sdmmc.p_api->erase(g_sdmmc.p_ctrl, start, count); Erase SDMMC sectors.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_ASSERTION	The pointer is NULL.
SSP_ERR_WRITE_PROTECTED	SD or MMC card is Write Protected.
SF_INFO_NOT_AVAILABLE	Information not available possibly because card has been removed or is defective.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_HW_LOCKED	The hardware lock has already been acquired.
SSP_ERR_TRANSFER_BUSY	The transfer is in progress.
SSP_ERR_CARD_NOT_READY	The card is not ready yet.
SSP_ERR_NOT_ENABLED	SDIO interrupt enable failed.
SSP_ERR_READ_FAILED	Read operation failed.

SSP_ERR_WRITE_FAILED	Write operation failed.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.43.3 SDMMC HAL Module Operational Overview

The following descriptions cover the operational functions and requirements when using the SDMMC HAL module:

Interrupt Configurations:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

Using SDIO with DTC:

- Access Interrupt
- SDIO Interrupt
- DTC Interrupt

Using SDIO with DMAC:

- Access Interrupt
- SDIO Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional and only available on MCU packages that have the SDHIn CD pin (n = channel number) available on the **Pins** tab of the Synergy Configuration tool.

Block Size Configuration

Block size configuration is provided for use with SDIO cards only. For SDIO cards, block size may be configured to 1-512 bytes. Block size must remain at the default 512 bytes for SD cards and e/MMC memory devices.

Card Detection Configuration

See *Card Detection Limitations* before using card detection in the SDMMC HAL driver. If card detection is not available or not desired for the application, Card Detection must be set to Not Used in the Properties for the r_sdmmc instance in the Synergy Configuration tool. To enable card detection, set Card Detection to CD Pin and Media Type to Card in the Properties for the r_sdmmc instance in the Synergy Configuration tool.

Access Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the access interrupt. This blocks other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

General Timing Notes

Several functions in this driver block. sdmmc_api_t::open() and sdmmc_api_t::erase() block until the entire operation is complete. sdmmc_api_t::read(), sdmmc_api_t::write(), sdmmc_api_t::readIo(), sdmmc_api_t::writeIo(), sdmmc_api_t::readIoExt(), and sdmmc_api_t::writeIoExt() block for a single command response cycle. In a multithreaded system, care should be taken to use this driver in a lower priority thread if other threads require a response time faster than the time this driver could block for during one of these function calls.

Timing Notes for Open

The sdmmc_api_t::open() API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1-bit and a bus speed of 400 kHz or less.

Timing Notes for Read, Write, ReadIoExtand WriteIoExt

The read and write media (sdmmc_api_t::read() and sdmmc_api_t::write()) and extended read and write SDIO APIs (sdmmc_api_t::readIoExt() and sdmmc_api_t::writeIoExt()) block until the response is received from the device. The data transfer operation is non-blocking and requires interrupts and a transfer instance, either DMAC or DTC. These APIs return SSP_SUCCESS to indicate that the initial operations have started successfully. The application must wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

Timing Notes for ReadIo and WriteIo

The SDIO sdmmc_api_t::readIo() and sdmmc_api_t::writeIo() APIs block until the response is received from the device. The read or write operation is complete when these APIs return.

Timing Notes for Erase

The sdmmc_api_t::erase() API blocks until the erase operation is complete. This may be several seconds or more depending on how many sectors are being erased.

SDIO Interrupts

Many SDIO cards use level interrupts, meaning the interrupt is not de-asserted until the interrupt is cleared on the device. In order to ensure SDIO interrupts are cleared appropriately, one of the following methods are recommended:

- Ensure the SDIO interrupt is cleared on the device before exiting the callback function with the event SDMMC_EVENT_SDIO. If this method is chosen and any SDIO API must be used in the interrupt, the access interrupt must be a higher priority than the SDIO interrupt.
- Disable the SDIO interrupt in the callback function with the event SDMMC_EVENT_SDIO using sdmmc_api_t::IoIntEnable(). Clear the SDIO interrupt elsewhere in the application, then re-enable SDIO interrupts if desired using sdmmc_api_t::IoIntEnable().

SDMMC HAL Module Important Operational Notes and Limitations

SDMMC HAL Module Operational Notes

SD HALA Compliance

When developing host devices that are compliant with the SD Specifications, the host must comply with the SD Host/Ancillary Product License Agreement (SD HALA).

SDMMC HAL Module Limitations

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdmmc driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred (see Access Interrupt Priority).

Card Detection Limitations

Card detection support in the r_sdmmc driver is limited. Card detection is only available after sdmmc_api_t::open() is complete, and open() cannot be completed unless a card is inserted. Card detection in the r_sdmmc driver is therefore only useful to detect card removal and reinsertion. After reinsertion is detected, the driver must be closed and reopened, and card detection will not be available until the reopen is complete. Card detection is available only on MCU packages that have the SDHIn CD pin (n = channel number) available on the Pins tab of the Synergy Configuration Tool. If the MCU does not have the SDHIn CD pin, or card detection is not desired for the application, card detection must be disabled in the Properties for the r_sdmmc instance in the Synergy Configuration Tool. An alternative to using the card detection feature of the r_sdmmc driver is to use an External IRQ instance and handle card detection at the application layer. If card detection is handled at the application layer, the sdmmc_api_t::open() should be called after card insertion is detected, and the sdmmc_api_t::close() should be called after card removal is detected.

Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.2.43.4 Including the SDMMC HAL Module in an Application

This section describes how to include the SDMMC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SD/MMC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SD/MMC Driver is g_sdmmc0. This name can be changed in the associated Properties window.)

SDMMC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sdmmc0 SD/MMC driver on r_sdmmc	Threads	**New Stack> Driver> Storage> SD/MMC Driver on r_sdmmc**

When the SDMMC HAL module on r_sdmmc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

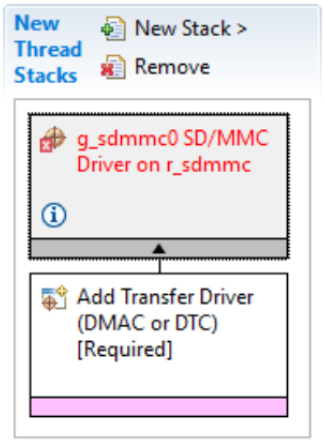


Figure 379: SDMMC HAL Module Stack

4.2.43.5 Configuring the SDMMC HAL Module

The SDMMC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

ConfigurationSettings for the SDMMC HAL Module on r_sdmmc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.

Name	g_sdmmc0	The name to be used for SDMMC module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	Select the channel.
Media Type	Embedded, Card Default: Embedded	Media is a card or an embedded device. This allows to firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 Bit, 4 Bits, 8 bits Default: 4 Bits	Data bus with as defined by hardware interface. (8 Bits for eMMC only).
Block Size	512	Select the media block size.
Card Detection	Not Used, CD Pin Default: CD Pin	Select the card detection method.
Write Protection	WP Pin, Not Used Default: WP Pin	Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.
Callback	NULL	(Required if not using FileX) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR.
Access Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Access interrupt priority selection.
Card Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Card interrupt priority selection.
DTC Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	DTC interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SDMMC HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer0	Module name.
Channel	0	Channel selection.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SDMMC HAL Module Clock Configuration

The SDMMC MCU peripheral (SDHI) uses the PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO or eMMC device obtained at media device initialization.

SDMMC HAL Module Pin Configuration

Use the ISDE pin configurator to configure the I/O pins for SDMMC peripheral (SDHI). The drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins:

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for SDMMC HAL Module on r_riic

Resource	ISDE Tab	Pin selection Sequence
SDHI	Pins	Select **Peripherals**> **Storage:SHDI**> **SDHI0**

Note

The selection sequence assumes SCI1 is the desired hardware target for the driver.

Pin Configuration Settings for the SDHI Peripheral

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, SD_MMC 1 bit SD_MMC 4 bit MMC 8 bit Default: Custom	Select mode as per application.
CLK	None, P413 Default: P413	Clock pin.
CMD	None, P412 Default: P412	Command pin.
DAT0-7	None, PXXX Default: PXXX	Data pin.
CD	None, P903 Default: P903	Card detection pin.
WP	None, P414 Default: P414	Card write protection pin.
SDA	Disabled	SDA Pin (when Simple I2C is used).
SCL	Disabled	SCL Pin (when Simple I2C is used).

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.43.6 Using the SDMMC HALModule in an Application

The steps for using the SDMMC HAL module in a typical application are:

1. Open the driver using the sdmmc_api_t::open() API.
2. Read data from the card using the sdmmc_api_t::read() API or write data to the card using sdmmc_api_t::write() API.
3. Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time the sdmmc_api_t::read() API or the sdmmc_api_t::write() API is called.

These common steps are illustrated in a typical operational flow diagram in the following figure:

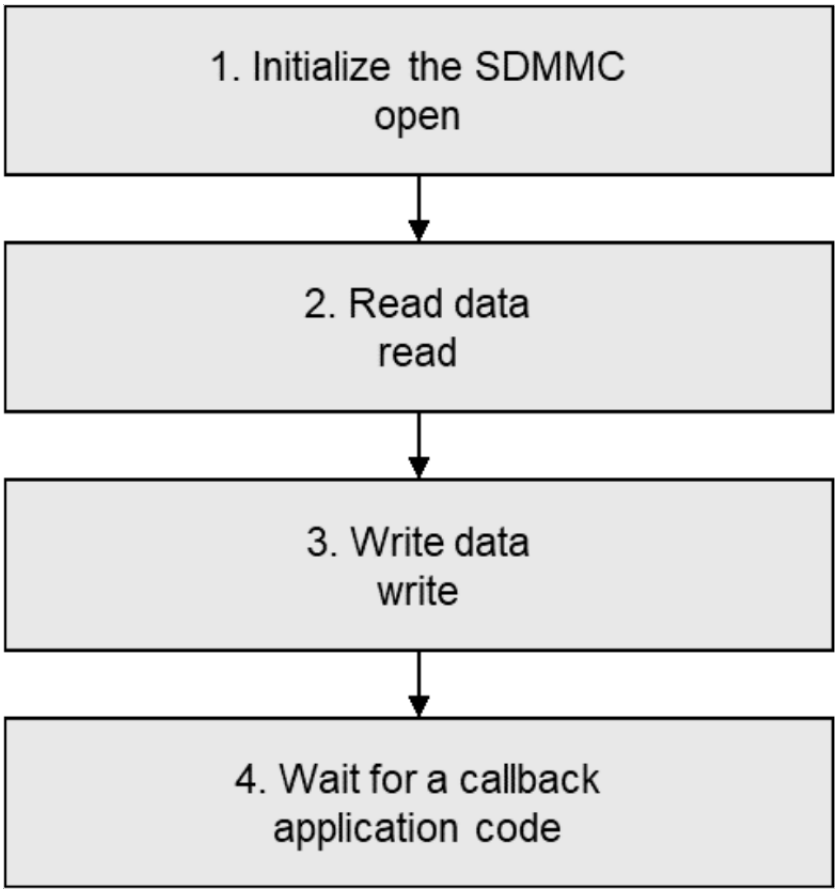


Figure 380: Flow Diagram of a Typical SDMMC HAL Module Application

The steps for using the SDMMC HAL module with an SDIO card in a typical application are:

1. Open the driver using the sdmmc_api_t::open() API.
2. Set or read back single registers using the sdmmc_api_t::readIo() API or the sdmmc_api_t::writeIo() API. The operation is complete immediately after these calls and there is no need to wait for a callback.
3. Set or read back multiple registers using the sdmmc_api_t::readIoExt() API or the sdmmc_api_t::writeIoExt() API. The block size can be changed using the sdmmc_api_t::control() API between calls if necessary.
4. Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time the sdmmc_api_t::readIoExt() API

or the sdmmc_api_t::writeIoExt() API is called.

5. If the card requests an interrupt, the callback is called with the event SDMMC_EVENT_SDIO. Handle the SDIO interrupt as described in the documentation for the card (see the SDIO Interrupts section of this documentation). SDIO interrupts from the card can be enabled or disabled at any time using the sdmmc_api_t::IoIntEnable() API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

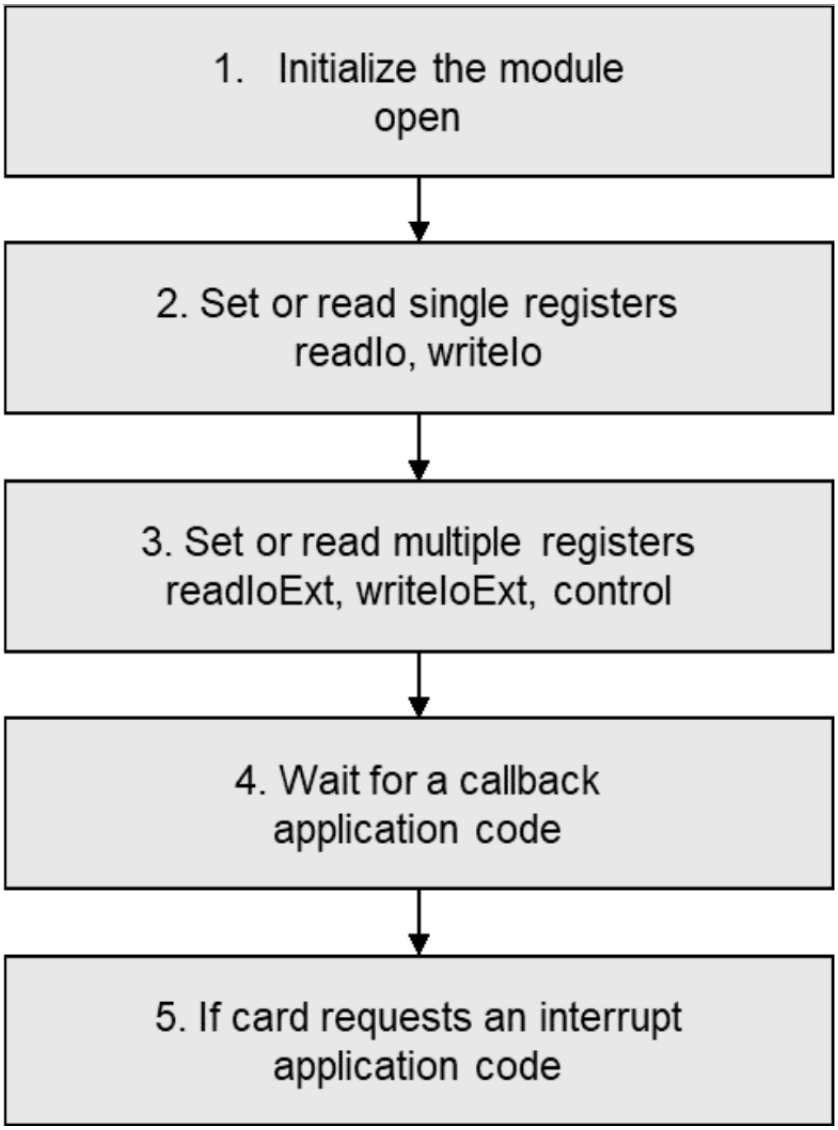


Figure 381: Flow Diagram of a Typical SDMMC HAL Module Application

4.2.44 Segment LCD Driver

4.2.44.1 SLCDC HAL Module Introduction

The Segment LCD HAL module provides a high-level API for Segment LCD applications and displays

and modifies data on a Segment LCD. The Segment LCD HAL module uses the Segment LCD Controller module on a Synergy MCU.

SLCDC HAL Module Features

- Internal voltage-boosting for the LCD driver voltage generator: select the capacitor split method or the external resistance division.
- Display bias: select the 1/2 bias method, 1/3 bias method, or 1/4 bias method.
- Time slice of the display: select static, 2-time slice, 3-time slice, 4-time slice, or 8-time slice.
- Display waveform: select waveform A or waveform B.
- Display data area: select A-pattern, B-pattern, or blinking. You can switch the display data area.
- Use the RTC periodic interrupt (PRD) to generate a blinking display with A-pattern and B-pattern.
- Adjust the reference voltage which is generated when operating the voltage boost circuit in 16 steps (contrast adjustment.)

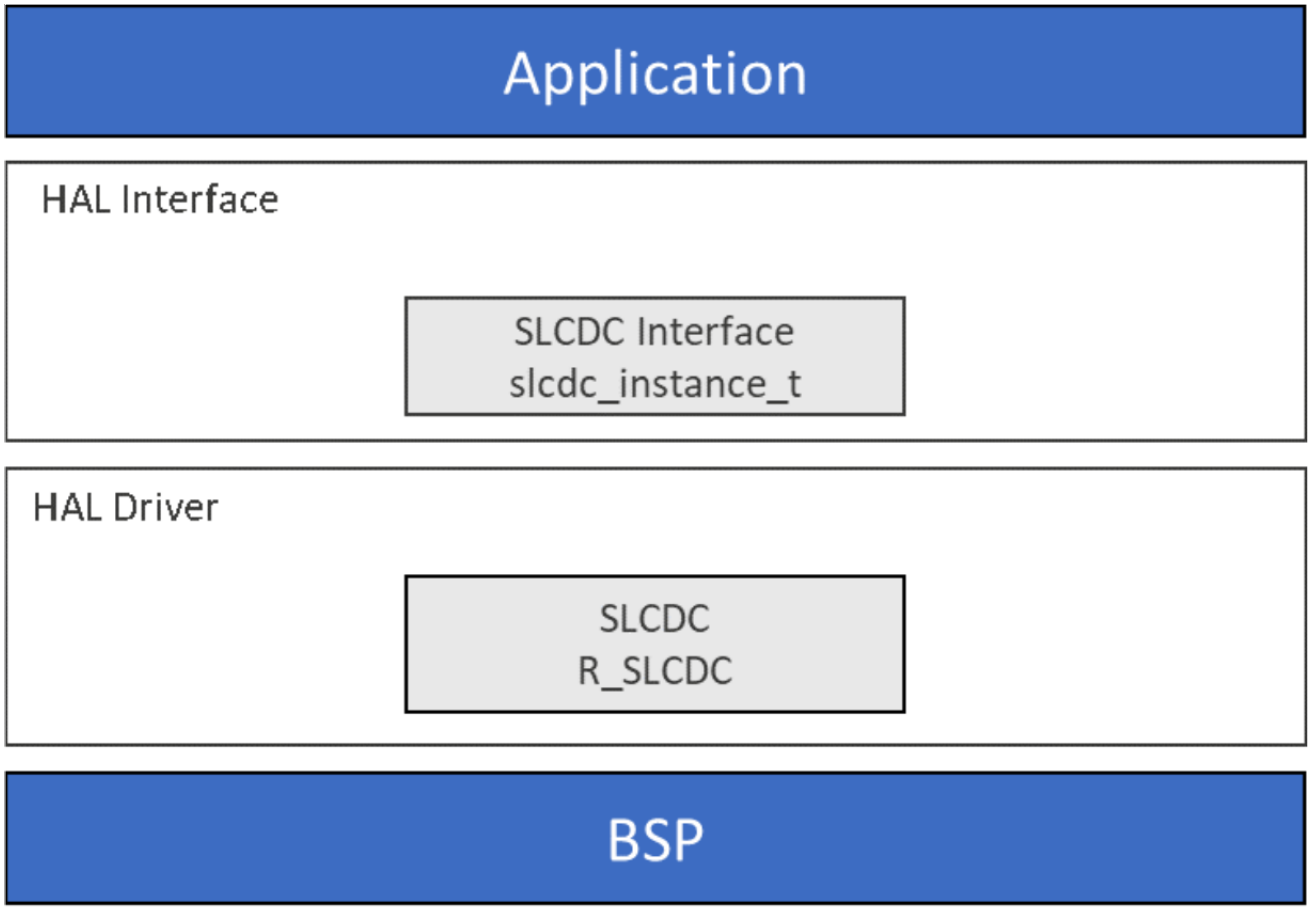


Figure 382: SLCDC HAL Module Block Diagram

SLCDC Hardware Support Details

The following hardware features are, or are not, supported by SSP for the SLCD.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Liquid crystal waveform (waveform A or B) selectable	Voltage Generator - Internal voltage boosting method, and external resistance division method	Voltage Generator - Capacitor split method	LCD blinking and display functions	Source Clock support: Main Clock Oscillator	Source Clock support: Sub Clock Oscillator
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	N/A	N/A	N/A	N/A	N/A	N/A
S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A	N/A	N/A

MCU Group	Source Clock support: Low speed Clock Oscillator	Source Clock support: High speed Clock Oscillator	Time slice modes – Static, 4	Time slice modes – 1, 2, 3, and 8	Bias method 2, 3, 4	Contrast adjustment
S124	N/A	N/A	N/A	N/A	N/A	N/A
S128	N/A	N/A	N/A	N/A	N/A	N/A
S1JA	N/A	N/A	N/A	N/A	N/A	N/A
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	N/A	N/A	N/A	N/A	N/A	N/A

S5D5	N/A	N/A	N/A	N/A	N/A	N/A
S5D9	N/A	N/A	N/A	N/A	N/A	N/A
S7G2	N/A	N/A	N/A	N/A	N/A	N/A

4.2.44.2 SLCDC HAL Module APIs Overview

The Segment LCD Controller HAL module defines APIs for functions such as opening, writing, starting, modifying and closing. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SLCDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_slcdc.p_api->open(g_slcdc.p_ctrl, g_slcdc.p_cfg); Open SLCD device.
write	g_slcdc.p_api->write(g_slcdc.p_ctrl, start_segment, &data, segment_count); Write data to SLCD segments. Specifies the initial display data. The data parameter is a pointer to an array of bytes consisting at least segment_count items, in which each byte is associated with one segment data register. When the number of time slices is static, 2, 3 or 4, the lower 4 bits of the data become an A-pattern area and the upper 4 bits become a B-pattern area. See slcdc_api_t::setdisplayArea for setting a display area.
modify	g_slcdc.p_api->modify(g_slcdc.p_ctrl, segment, data_mask, data); Rewrite data in the SLCD segment. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is. Specifies the data to rewrite.
start	g_slcdc.p_api->start(g_slcdc.p_ctrl); Enable display on the SLCD. Displays the specified data on the LCD. Before that data should be written to the segments.
stop	g_slcdc.p_api->stop(g_slcdc.p_ctrl); Disable display on the SLCD. Stops displaying data on the SLCD.
contrastIncrease	g_slcdc.p_api->contrastIncrease(g_slcdc.p_ctrl); Increase the display contrast. Increase by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator.

contrastDecrease	g_slcdc.p_api->contrastDecrease(g_slcdc.p_ctrl); Decrease the display contrast. Decrease by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator.
setdisplayArea	g_slcdc.p_api->setdisplayArea(g_slcdc.p_ctrl, display_area); Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to set blink on where A-pattern and B-pattern area data will be alternately displayed. When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1. Open RTC 2. Set Periodic interrupt request, ½ second 3. Start RTC counter 4. Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual: Microcontrollers for the detailed procedure.
close	g_slcdc.p_api->close(g_slcdc.p_ctrl); Close display device.
versionGet	g_slcdc.p_api->versionGet(&version); Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Invalid Argument.
SSP_ERR_HW_LOCKED	SLCDC resource is locked.
SSP_ERR_NOT_OPEN	Device is not opened or initialized.
SSP_ERR_UNSUPPORTED	Unsupported operation.
SSP_ERR_NOT_ENABLED	RTC not enabled for blink operation.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.44.3 SLCDC HAL Module Operational Overview

This module uses the Segment LCD controller (SLCDC) to display data on a Segment LCD. The driver initializes the LCD for displaying data and configures the drive-voltage generator, display waveform, number of time slices and bias methods to drive the LCD. This module provides functions to display data to a specified set of segments, to modify existing segment data, to enable and disable display, to set the display area and to adjust the contrast. The contents displayed on the LCD can be changed by changing the contents of the LCD display data registers.

SLCDC HAL Module Important Operational Notes and Limitations

SLCDC HAL Module Operational Notes

- This driver is a HAL driver and has no dependencies with the ThreadX RTOS. You can add the Segment LCD HAL module to a thread in the ThreadX RTOS if it is desirable.
- To write to a sequence of segments, give the start segment number and number of segments to be written in the slcdc_api_t::write API.

SLCDC HAL Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.44.4 Including the SLCDC HAL Module in an Application

This section describes how to include the SLCDC HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Segment LCD Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Segment LCD Driver is g_timer0. This name can be changed in the associated Properties window.)

SLCDC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_slcdc0 Segment LCD Driver on r_slcdc	Threads	**New Stack> Driver> Graphics> Segment LCD Driver on r_slcdc**

When the Segment LCD Driver on r_slcdc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

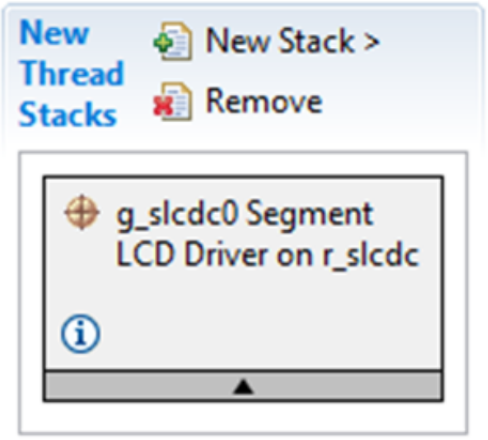


Figure 383: SLCDC HAL Module Stack

4.2.44.5 Configuring the SLCDC HAL Module

The SLCDC HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SLCDC HAL Module on r_slcdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Select if extra code will be added to check parameter values.
Name	g_slcdc0	Module name.
Slcdc Clock	Clock Loco, Clock Sosc, Clock Mosc, Clock Hoco Default: Clock Hoco	SLCD clock source (LCDSCKSEL).

Slcdc Clock Divisor	Clk Divisor Loco 4, Clk Divisor Loco 8, Clk Divisor Loco 16, Clk Divisor Loco 32, Clk Divisor Loco 64, Clk Divisor Loco 128, Clk Divior Loco 256, Clk Divisor Loco 512, Clk Divisor Loco 1024, Clk Divisor Hoco 256, Clk Divisor Hoco 512, Clk Divisor Hoco 1024, Clk Divisor Hoco 2048, Clk Divisor Hoco 4096, Clk Divisor Hoco 8192, Clk Divisor Hoco 16384, Clk Divisor Hoco 32768, Clk Divisor Hoco 65536, Clk Divisor Hoco 131072, Clk Divisor Hoco 262144, Clk Divisor Hoco 524288 Default: Clk Divisor Hoco 16384	LCD clock setting (LCDC0), clock divisor.
Bias Method	Bias 2, Bias 3, Bias 4 Default: Bias 2	LCD display bias method select (LBAS bit).
Time Slice	Static, Slice 2, Slice 3, Slice 4, Slice 8 Default: Static	Time slice of LCD display select (LDTY bit).
Wave Form	Wave A, Wave B Default: Wave A	LCD display waveform select (LWAVE bit).
Slcdc Drive Voltage Generator	External resistance division, Internal voltage boosting, Capacitor split Default: External resistance division	LCD Drive Voltage Generator Select (MDSTET bit).

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SLCDC HAL Module Clock Configuration

Configure the SLCDC Hal module clock in the properties window of the g_slcdc driver. The operating clock of the SLCDC HAL Module is specified by the SLCDC Clock and SLCDC Clock Divisor settings in the Properties window. The Segment LCD HAL module-source clock can be configured as the Main (MOSC), HOCO (High-speed Clock Oscillator), LOCO (Low-speed Clock Oscillator) or Sub-clock (SOSC) using the ISDE configurator. For HOCO and LOCO settings, several clock divisors are available.

SLCDC HAL Module Pin Configuration

The SLCDC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates

the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note

For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for the SLCDC HAL Module on r_slcdc

Resource	ISDE Tab	Pin Selection Sequence
SLCDC	Pins	Select **Peripherals**> **Graphics: SLCDC**> **SLCDC0**

Note

The selection sequence assumes SLCDC0 is the desired hardware target for the driver.

Pin Configuration Settings for SLCDC HAL Module

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, Static, 2x Slice, 3x Slice, 4x Slice, 8x Slice Default: Custom	Select operation mode enable or disable.
CAPH	None, P111 Default: None	Capacitor connection pin.
CAPL	None, P112 Default: P112	Capacitor connection pin.
COM0:3	None, Pn Default: P104:107	Common pins.
COM4:7	None, Pn Default: None	Common pins.
VL1:4	None, Pn Default: P100:103	Power supply pins.
SEG00:02, SEG06:07, SEG16:17, SEG21:25, SEG46:51	None, Pn Default: None	Segment pins.
SEG03	None, P303 Default: P303	Segment pin.

SEG04:05	None, Pn Default: P314:315	Segment pins.
SEG08	None, P902 Default: P902	Segment pin.
SEG09:15	None, Pn Default: P312:306	Segment pins.
SEG18:19	None, Pn Default: P808:809	Segment pins.
SEG20	None, P313 Default: P313	Segment pin.
SEG26:27	None, Pn Default: P806:807	Segment pins.
SEG28:34	None, Pn Default: P608:614	Segment pins.
SEG35:41	None, Pn Default: P606:600	Segment pins.
SEG42:43	None, Pn Default: P805:804	Segment pins.
SEG44:45	None, Pn Default: P800:801	Segment pins.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.44.6 Using the SLCDC HAL Module in an Application

The typical steps in using the SLCDC HAL module in an application are:

1. Initialize the SLCD HAL module using the slcdc_api_t::open API.
2. Write a sequence of segments using the slcdc_api_t::write API.
3. Modify the content of the segment data using the slcdc_api_t::modify API if needed.
4. Change the display area or blinking display using the slcdc_api_t::setdisplayArea API.
5. Enable the display by using the slcdc_api_t::start API.
6. Adjust contrast using the slcdc_api_t::contrastIncrease or slcdc_api_t::contrastDecrease APIs.
7. Disable the display by using the slcdc_api_t::stop API.
8. Close the driver using the slcdc_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

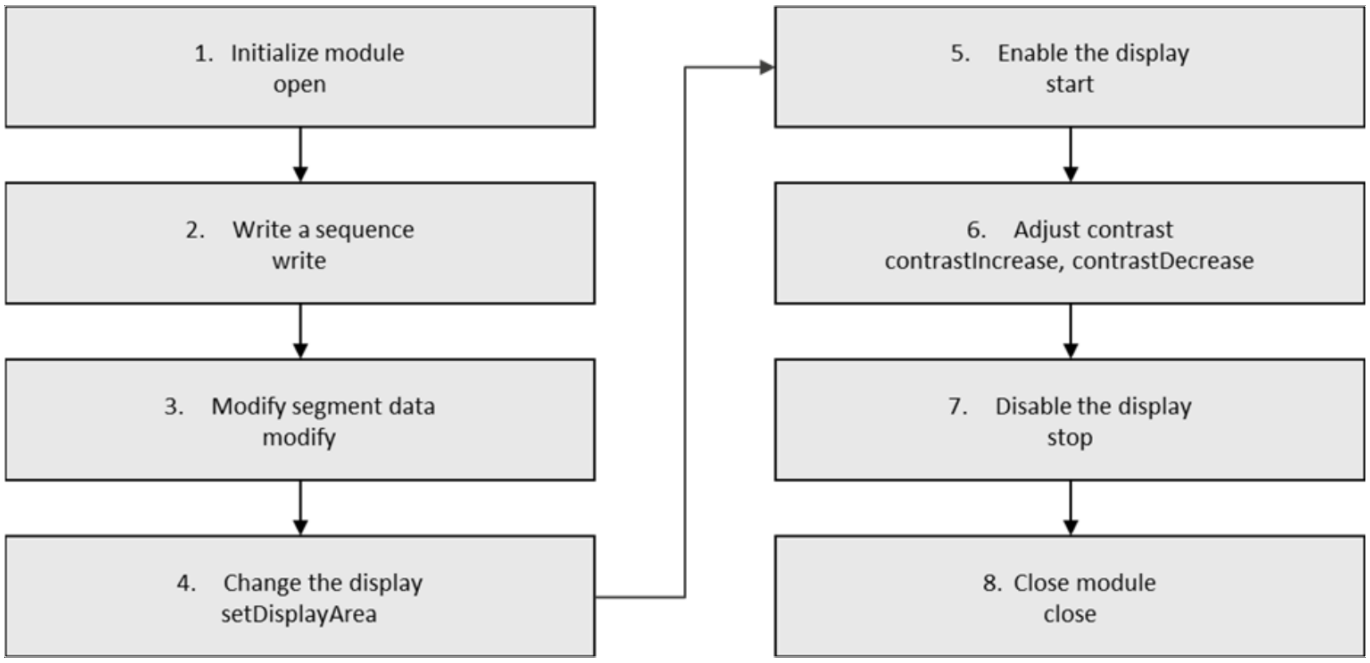


Figure 384: Flow Diagram of a Typical SLCDC HAL Module Application

4.2.45 SCI SPI Driver

4.2.45.1 SCI SPI HAL Module Introduction

The SCI SPI HAL module provides a high-level API for master/slave-based industry standard SPI serial communications and configures and uses the SCI (Serial Communications Interface) peripheral on a Synergy MCU. A user-defined callback can be created to signal when the SPI has transmitted data, aborted a data transfer or detected an error condition.

The SCI SPI HAL module is enabled with a data transfer function support by incorporating the Data Transfer Controller module of the MCU. This performs SPI transfers through the DTC without intervention of the CPU.

SCI SPI HAL Module Features

The SCI SPI HAL module supports the configuration and control of the SPI functions on the Synergy MCU. Key features include the following:

- Driver initialization
- Serial communication through SPI operation using 8-bit data transfers
- Configurable among four clock phase and clock polarity settings
- Support for callbacks. The callback functions are called with the following events:
    - Transfer aborted
    - Transfer complete
    - Over run error

- SPI communication in master and slave mode.

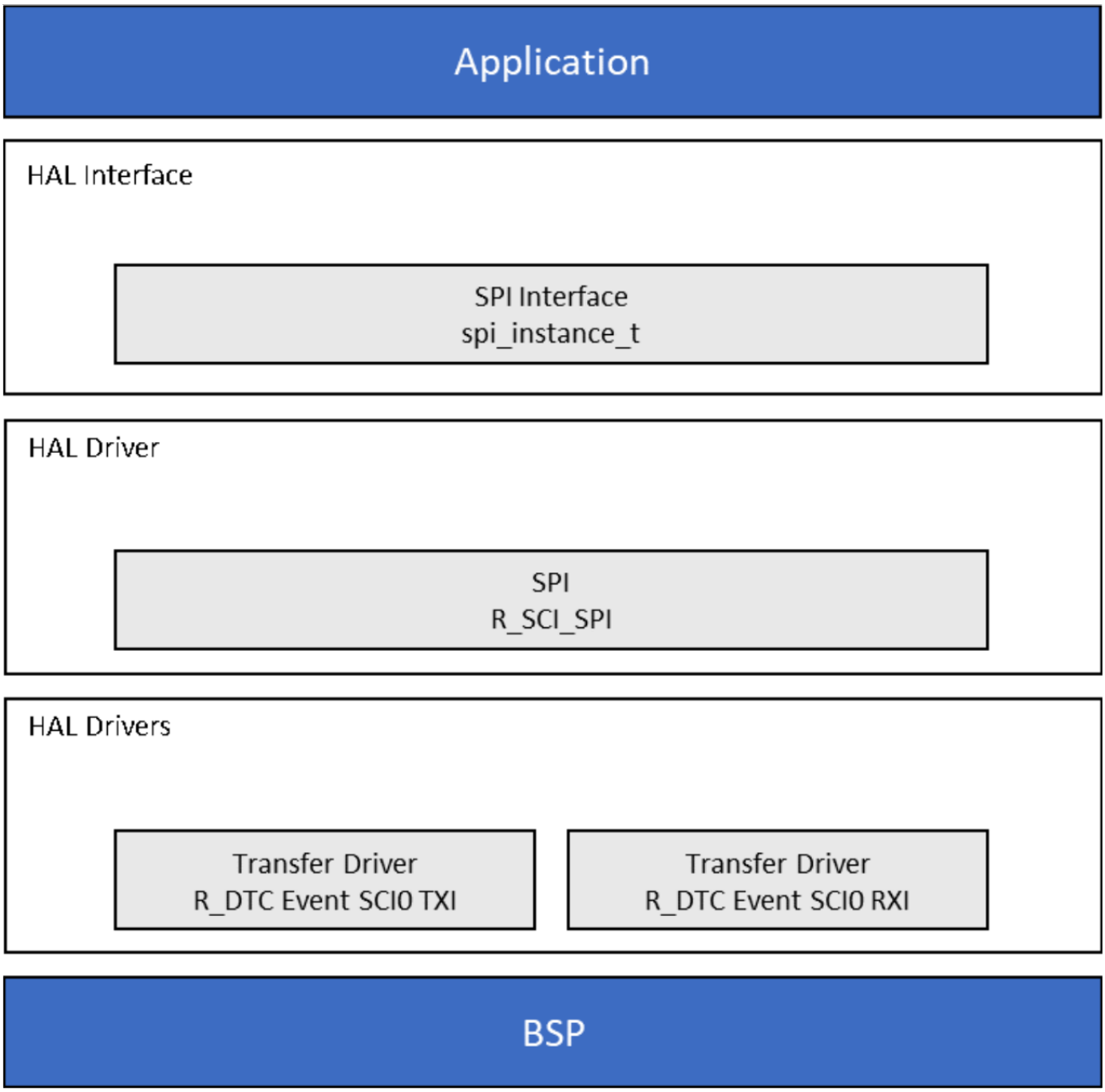


Figure 385: SCI SPI HAL Module Block Diagram

SCI SPI Hardware Support Details

The following hardware features are, or are not, supported by SSP for the SCI_SPI:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)

N/A				Not supported by MCU		
MCU Group	Master Out/Slave In (MOSI)	Master In/Slave Out (MISO)	SSL Slave Select	SPI Operation (4-wire method)	Clock Synchronous Operation (3-wire method)	Full Duplex or Transmit Only selectable
S124	✓	✓	GPIO	N/A	✓	✓
S128	✓	✓	GPIO	N/A	✓	✓
S1JA	✓	✓	GPIO	N/A	✓	✓
S3A1	✓	✓	GPIO	N/A	✓	✓
S3A3	✓	✓	GPIO	N/A	✓	✓
S3A6	✓	✓	GPIO	N/A	✓	✓
S3A7	✓	✓	GPIO	N/A	✓	✓
S5D3	✓	✓	GPIO	N/A	✓	✓
S5D5	✓	✓	GPIO	N/A	✓	✓
S5D9	✓	✓	GPIO	N/A	✓	✓
S7G2	✓	✓	GPIO	N/A	✓	✓
MCU Group	Overrun error detection	Data format transfer bit length	Master or Slave	Clock source Internal/external	Configurable phase and polarity	Interrupt sources support
S124	✓	8-bit	✓	Internal	✓	✓
S128	✓	8-bit	✓	Internal	✓	✓
S1JA	✓	8-bit	✓	Internal	✓	✓
S3A1	✓	8-bit	✓	Internal	✓	✓
S3A3	✓	8-bit	✓	Internal	✓	✓
S3A6	✓	8-bit	✓	Internal	✓	✓
S3A7	✓	8-bit	✓	Internal	✓	✓
S5D3	✓	8-bit	✓	Internal	✓	✓
S5D5	✓	8-bit	✓	Internal	✓	✓
S5D9	✓	8-bit	✓	Internal	✓	✓
S7G2	✓	8-bit	✓	Internal	✓	✓

4.2.45.2 SCI SPI HAL Module APIs Overview

The SPI defines APIs for opening and closing the SCI peripheral and transmitting and receiving data. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SCI SPI HAL Module API Summary

Function Name	Example API Call and Description
.open	g_spi.p_api ->open(g_spi.p_cntl, g_spi.p_cfg); Open a designated SPI device.
.read	g_spi.p_api->read(g_spi.p_ctrl, dst16, length, SPI_BIT_WIDTH_16_BITS); Receive data from SPI device.
.write	g_spi.p_api->write (g_spi.p_ctrl, source, length, SPI_BIT_WIDTH_8_BITS); Transmit data to SPI device
.writeRead	g_spi.p_api ->writeRead (g_spi.p_cntl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Simultaneously transmits data to an SPI device, while receiving data from an SPI device (full duplex). The writeRead API fetches the mutex object, handles SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to complete the data transfer.
.close	g_spi.p_api->close(g_spi,p_ctrl) Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
.versionGet	g_spi.p_api ->versionGet (&version); Get the version information of the underlying driver.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_IN_USE	Attempted to open an already open device instance OR Another transfer was in progress.
SSP_ERR_INVALID_POINTER	p_version is NULL.
SSP_INVALID_ARGUMENT	Channel number invalid.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.

SSP_ERR_CH_NOT_OPEN	The channel has not been opened. Open channel first.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.45.3 SCI SPI HAL Module Operational Overview

The SCI SPI HAL module provides the ability to configure and use the SPI capabilities of the Synergy MCU. The SCI SPI HAL module enables communication with a peripheral device using the SPI communications protocol. After opening the SCI SPI HAL module instance, the SCI SPI module handle is used to perform various transfer operations. The device control handle will be used within the API calls to indicate the specific SCI SPI device to communicate with.

The SCI SPI HAL module allows the user to:

- Initialize the module.
- Serial Communication through SPI operation. Read from and write to (and simultaneous read/write – full duplex) a SPI device – performed by calling the spi_api_t::read, spi_api_t::write and spi_api_t::writeRead APIs.

The Driver also provides support for callbacks. The callback functions are called with the following events:

- Transfer aborted
- Transfer complete
- Overrun error

The SCI SPI module supports only 8-bit data transfer operations. The SCI SPI module uses GPIO pins configured as chip selects.

Clock Settings

The SCI SPI uses PCLKA as its clock source. You can set the PCLKA frequency using the clock configurator in e^2 studio or the CGC Interface at run-time.

I/O Port Settings

To use with the SPI, the I/O port pin(s) used as output pins must be configured as SCI SPI peripheral pins in the pin configurator. For external chip select, configure Chip select pin as GPIO output.

SCI SPI Interrupts

To enable the interrupts of the SCI SPI, highlight the driver module and set the priority of the SCI RXI, TXI, TEI and ERI interrupts on the **Threads** tab of the Project Configurator in e^2 studio: Configuring Interrupts.

This sets the corresponding interrupts in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

Note

Setting the interrupts to different priority levels could result in improper operation.

SCI SPI HAL Module Important Operational Notes and Limitations

SCI SPI HAL Module Operational Notes

- Chip select outputs are supported using GPIOs
- The SCI SPI HAL module uses only 8-bit data transfers.
- Setting the interrupts to different priority levels could result in improper operation.
- The SCI SPI HAL module is enabled with a data transfer support by incorporating the Data Transfer Controller module of the MCU. This performs an SPI transfer through the DTC without intervention of the CPU. The DTC transfer is enabled by default; the user has to remove it from the configurator for an IRQ mode transfer.

SCI SPI HAL Module Limitations

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.45.4 Including the SCI SPI HAL Module in an Application

This section describes how to include the SPI HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SPI Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SPI Driver is g_spi0. This name can be changed in the associated Properties window.)

SCI SPI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_spi0 SPI Driver on r_sci_spi	Threads	**New Stack> Driver> Connectivity> SPI Driver on r_sci_spi**

When the SPI Driver on r_sci_spi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

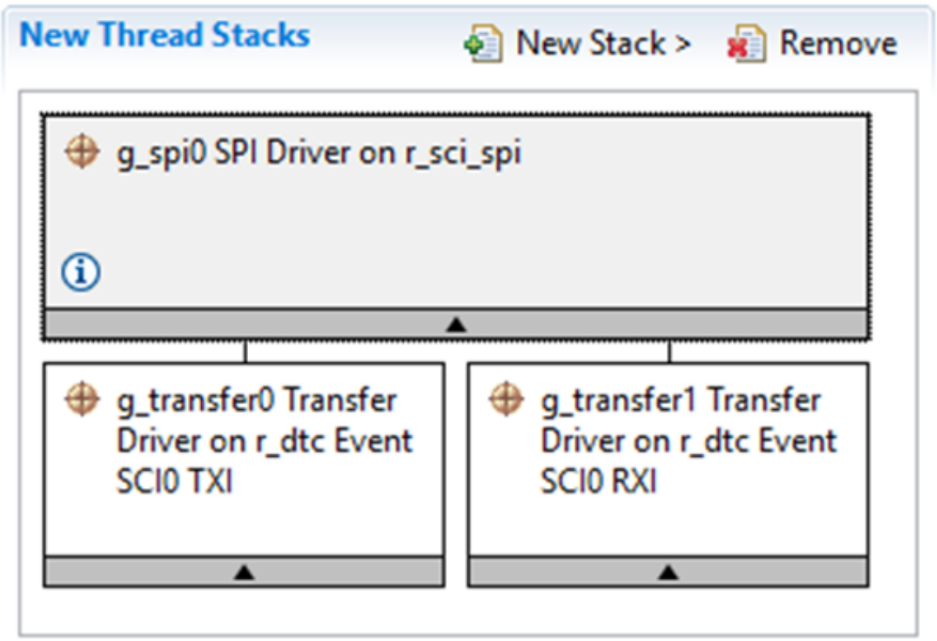


Figure 386: SPI HAL Module Stack

4.2.45.5 Configuring the SCI SPI HAL Module

The SPI HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SCI SPI HAL Module on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.

Operating Mode	Master, Slave Default : Master	Configure as a Master or Slave device.
Clock Phase	Data sampling on odd edge, data variation on even edge/Data sampling on even edge, data variation on odd edge Default: Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.
Clock Polarity	Low when idle, High when idle Default: Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first.
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable.
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Error interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SCI SPI HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency seleciton.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC software event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SCI SPI HAL Module Clock Configuration

The SCI peripheral is clocked via the Peripheral Clock A (PCLKA.) The clock frequencies are configurable in the ISDE by using the Clocks tab in the configurator. Invalid selections are indicated in red when selected. Ensure that desired SPI bitrate can be achieved with the stated value of PCLKA. The ISDE will not be indicated if the specified bitrate is not achievable. At run time, the SPI HAL module will attempt to configure the SCI peripherals to the correct bitrate and will return an error if the desired bitrate cannot be set. The bitrate is calculated via the equations in the following table. If the result of the equation (N) is in the range of 0 to 255, then the bit rate can be achieved.

Baud Rate Calculation Equations

SPI HAL	Bitrate calculation	Description
SPI on SCI	$N = \frac{PCLKA\ (MHz)}{8 * 2^{(2n-1)} * \left(256/M\right) * B} - 1$	N = Peripheral register value. This must be in the range of 0 to 255 PLCKA = value of PLCKA in MHz n = 0, 1, 2 or 3 M = Bit Rate Modulation Index 128 < M < 256 *If the Bit Rate Modulation is disabled, then M=256* B = Desired Bit Rate

SCI SPI HAL Module Pin Configuration

The SCI peripheral use pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the SPI pins:

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for SCI SPI HAL Module on r_sci_spi

Resource	ISDE Tab	Pin Selection Sequence
SPI on SCI	Pins	Select **Peripherals**> **Connectivity:SCI**> **SCIx,** where x is the required SCI peripheral channel.

Note

The selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for the SCI SPI HAL Module on r_sci_spi

Pin Configuration Property	Value	Description
Pin Group Selection	Mixed, _A only, _B only	Synergy devices support peripheral functionality via multiple pins location, identified by _A, _B. Selecting Mixed allows the user to select any combination of locations (_A and _B). Selecting _A allows the user to select only _A locations. Selecting _B allows the user to select only _B locations.
Operation Mode	Disabled Custom Asynchronous UART Simple SPI Simple I2C Synchronous UART Smart Card	Set the operating mode to: Simple SPI.
TXD_MOSI	None, P411, P101	Specify the port pin to be used as MOSI.
RXD_MISO	None, P410, P100	Specify the port pin to be used as MISO.
SCK	None, P412, P102	Specify the port pin to be used as CLK.
CTS_RTS_SS	None, P413, P103	Specify the port pin to be used as SS.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.*

4.2.45.6 Using the SCI SPI HAL Module in an Application

The steps in using the SCI SPI HAL module in a typical application are:

Note

The spi_api_t::open API function must be called first. The rest of the calls may be used in any order depending on the application requirements:

1. Initialize an SPI instance using the spi_api_t::open API function. (g_spi.p_api->open (g_spi.p_ctrl, g_spi.p_cfg) where p_ctrl and p_cfg are the instances of control and configuration structures autogenerated after the configuration step).
2. Initiate a write to a slave device using the spi_api_t::write API function. (g_spi.p_api->write (g_spi.p_ctrl, source, length, SPI_BIT_WIDTH_8_BITS); where g_spi.p_ctrl is the same control instance that was used in the spi_api_t::open call).

3. Initiate a read from a slave device using the spi_api_t::read API function. (g_spi.p_api->read(g_spi.p_ctrl, dst, length, SPI_BIT_WIDTH_8_BITS); where g_spi.p_ctrl is the same control instance that was used in the spi_api_t::open call).
4. Initiate a data transfer in both directions with a slave device using the spi_api_t::writeRead API function. (g_spi.p_api->writeRead(g_spi.p_ctrl, source, s_length, destination, d_length, SPI_BIT_WIDTH_8_BITS); where g_spi.p_ctrl is the same control instance that was used in the spi_api_t::open call).
5. Use the spi_api_t::close API function to close the instance. (g_spi.p_api->close(g_spi,p_ctrl) where g_spi.p_ctrl is the same control structure that was used in the spi_api_t::open call).

These common steps are illustrated in a typical operational flow diagram in the following figure:

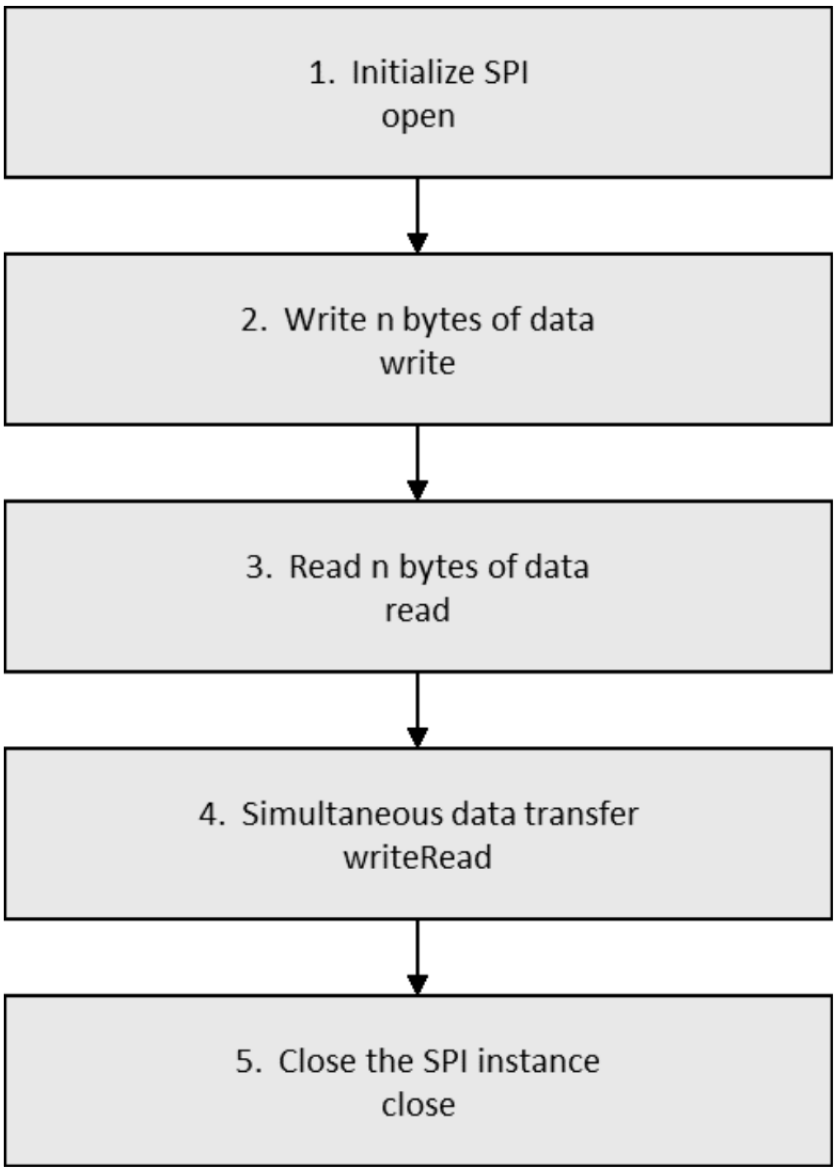


Figure 387: Flow Diagram of a Typical SPI HAL Module Application

4.2.46 SPI Driver

4.2.46.1 RSPI HAL Module Introduction

The RSPI HAL module provides a high-level API for serial communication using the SPI protocol. The module supports the SPI (formerly known as RSPI) peripheral available on the Synergy microcontroller hardware. The RSPI HAL module supports standard SPI master and Slave mode communications functions. Callbacks are provided for transfer events. The RSPI HAL module is enabled with data transfer support by incorporating the data transfer controller module of the MCU; this performs SPI transfers through the DTC without requiring interrupt processing for each frame.

RSPI HAL Module Features

- Initialization of the driver
- SPI transfer functions:
    - Allows serial communication through the SPI operation using the four-wire method
    - Capable of serial communication in master and slave modes
    - Switching the polarity of the serial transfer clock
    - Switching the phase of the serial transfer clock
- Data Format
    - MSB-first/LSB-first selectable
    - Transfer bit length is selectable as 8, 16 and 32 bits
    - 16-bit and 32-bit byte swapping for both received and transmitted data register
- Error Detection
    - Mode fault detection
    - Overrun error detection
    - Parity error detection
- SSL control functions
    - External hardware slave select can be used in master mode
- Interrupts
    - RSPI receive interrupt (receive buffer full)
    - RSPI transmit interrupt (transmit buffer empty)
    - RSPI error interrupt (mode fault, overrun and parity error)
- Delays
    - Add SPI clock delay
    - Add slave select negation delay
    - Add next-access delay

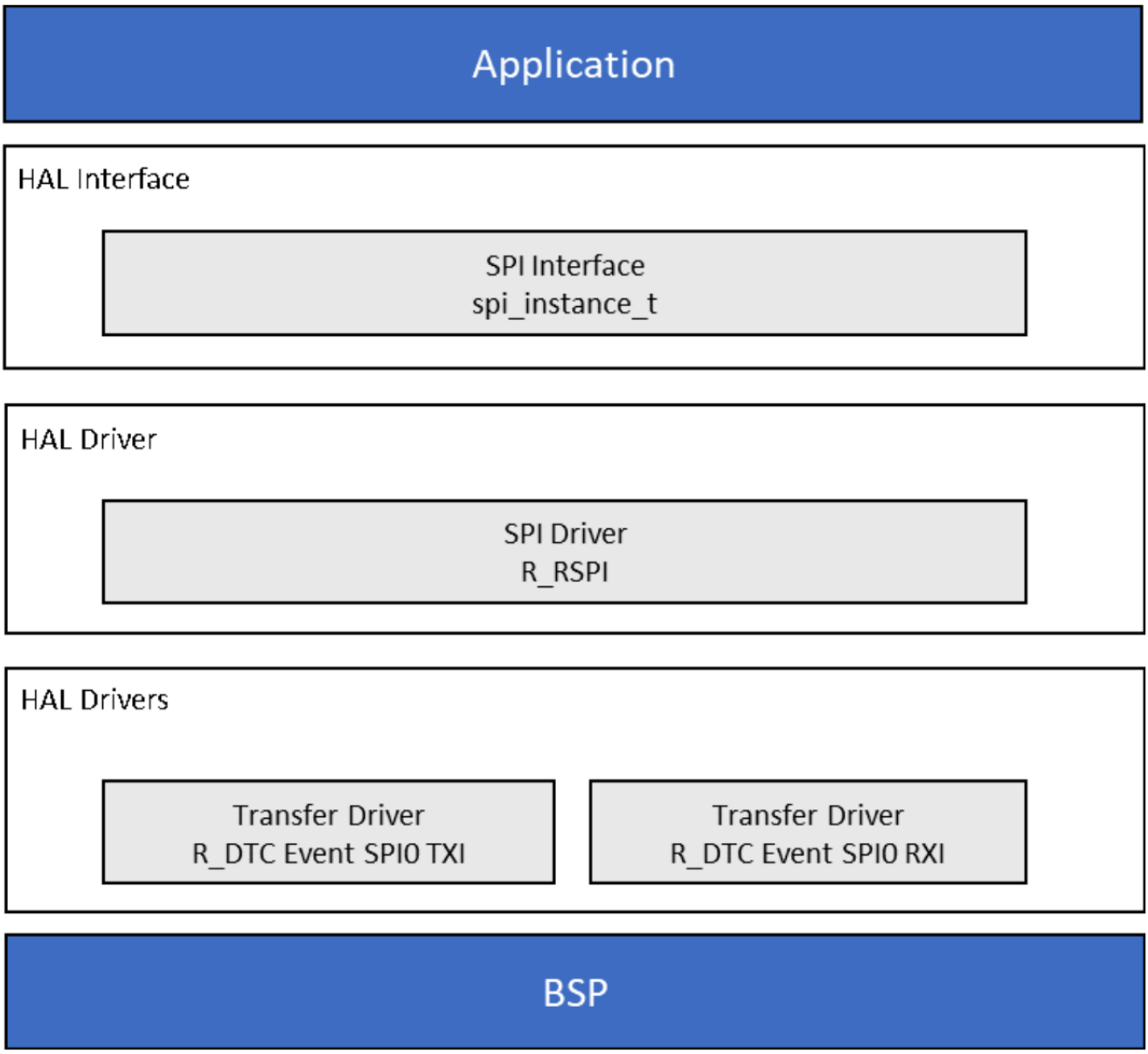


Figure 388: RSPI HAL Module Block Diagram

RSPI Hardware Support Details

The following hardware features are, or are not, supported by SSP for the RSPI.

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	SPI Operation mode	Clock Syn mode	Full-duplex or transmit-only can be selected	Switching of the polarity and phase	Master and Slave mode	MSB first/LSB first selectable
S124	✓	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Transfer length - 8/16/32	Up to four frames can be transferred in one round	Bit rate configuration	Double buffer configuration	Error detection	SSL control function
S124	✓	☒	✓	☒	✓	✓
S128	✓	☒	✓	☒	✓	✓
S1JA	✓	☒	✓	☒	✓	✓
S3A1	✓	☒	✓	☒	✓	✓
S3A3	✓	☒	✓	☒	✓	✓
S3A6	✓	☒	✓	☒	✓	✓
S3A7	✓	☒	✓	☒	✓	✓
S5D3	✓	☒	✓	☒	✓	✓
S5D5	✓	☒	✓	☒	✓	✓
S5D9	✓	☒	✓	☒	✓	✓
S7G2	✓	☒	✓	☒	✓	✓

MCU Group	Transfer of up to eight commands	All Interrupt sources	DTC Support	Event link function through ELC HAL driver	Function for switching between CMOS output and open drain output	Loopback mode

S124	☒	✓	✓32-bit transfer	☒	✓	✓
S128	☒	✓	✓32-bit transfer	☒	✓	✓
S1JA	☒	✓	✓32-bit transfer	☒	✓	✓
S3A1	☒	✓	✓32-bit transfer	☒	✓	✓
S3A3	☒	✓	✓32-bit transfer	☒	✓	✓
S3A6	☒	✓	✓32-bit transfer	☒	✓	✓
S3A7	☒	✓	✓32-bit transfer	☒	✓	✓
S5D3	☒	✓	✓32-bit transfer	☒	✓	✓
S5D5	☒	✓	✓32-bit transfer	☒	✓	✓
S5D9	☒	✓	✓32-bit transfer	☒	✓	✓
S7G2	☒	✓	✓32-bit transfer	☒	✓	✓

MCU Group	Module Stop Function	Multi-master mode
S124	☒	☒
S128	☒	☒
S1JA	☒	☒
S3A1	☒	☒
S3A3	☒	☒
S3A6	☒	☒
S3A7	☒	☒
S5D3	☒	☒
S5D5	☒	☒
S5D9	☒	☒
S7G2	☒	☒

4.2.46.2 RSPI HAL Module APIs Overview

The RSPI HAL module defines API functions for opening, closing, reading, writing and other useful functions. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

RSPI HAL Module API Summary

Function Name	Example API Call and Description
.open	g_spi.p_api ->open(g_spi.p_cntl, g_spi.p_cfg); Open a designated SPI device.
.read	g_spi.p_api->read(g_spi.p_ctrl, dst16, length, SPI_BIT_WIDTH_16_BITS); Receive data from SPI device.
.write	g_spi.p_api->write (g_spi.p_ctrl, source, length, SPI_BIT_WIDTH_8_BITS); Transmit data to SPI device
.writeRead	g_spi.p_api ->writeRead (g_spi.p_cntl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Simultaneously transmits data to an SPI device, while receiving data from an SPI device (full duplex). The writeRead API fetches the mutex object, handles SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to complete the data transfer.
.close	g_spi.p_api->close(g_spi,p_ctrl) Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
.versionGet	g_spi.p_api ->versionGet (&version); Get the version information of the underlying driver.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully
SSP_ERR_INVALID_MODE	Invalid mode
SSP_ERR_INVALID_CHANNEL	Invalid channel
SSP_ERR_IN_USE	In-use error
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_QUEUE_UNAVAILABLE	Queue unavailable

SSP_ERR_INVALID_POINTER	Invalid pointer
SSP_ERR_INTERNAL	Internal error
SSP_ERR_TRANSFER_ABORTED	Transfer aborted
SSP_ERR_MODE_FAULT	Mode fault
SSP_ERR_READ_OVF	Read overflow
SSP_ERR_PARITY	Parity error
SSP_ERR_OVERRUN	Overrun error
SSP_ERR_UNDEF	Unknown error
SSP_ERR_TIMEOUT	Timeout error
SSP_ERR_NOT_OPEN	Device not opened

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.46.3 RSPI HAL Module Operational Overview

The RSPI HAL module enables communication with a peripheral device using the SPI communications protocol. After opening the module instance, the module handle is used to perform various transfer operations. The device control handle will be used within the API calls to indicate the specific SPI device with which to communicate.

The Driver allows the application program to:

- Initialize the driver.
- Implement serial communication through SPI operation.

The module also provides support for callbacks. The callback functions are called with the following events spi_event_t

- Transfer aborted
- Transfer complete
- Mode fault
- Error events

The RSPI HAL module supports 8, 16 and 32-bit data transfers. The module supports GPIO pins configured as chip selects. In addition, the SPI peripheral supports dedicated chip select signal, SSL. When the SSL pin is enabled in the SPI peripheral, chip select handling is performed by the hardware.

Clock settings:

The SPI peripheral uses the PCLKA as its clock source. You can set the PCLKA frequency using the clock configurator in e^2 studio or the CGC Interface at run-time.

Note

For S1 devices the SPI peripheral clock source is PCLKB.

IO Port settings:

To use with the SPI peripheral, the I/O port pin(s) used as output pins must be configured as SPI peripheral pins in the pin configurator. If you are using an external chip select, configure Chip select pin as GPIO output.

Extended configuration:

A number of extended hardware specific configurations are present for SPI Driver.

Note

All extended hardware specific configuration parameters are set in the extended driver configuration structure spi_on_rspi_cfg_t.

RSPI HAL Module Important Operational Notes and Limitations

RSPI HAL Module Operational Notes

While configuring the RSPI HAL drivers, setting the interrupts to different priority levels could result in improper operation.

The module is enabled with a data transfer support by incorporating the Data Transfer Controller module of the MCU. This performs SPI transfers through the DTC without the intervention of the CPU.

In the application, data transfers over the DTC are used in the same way as normal SPI transfers. To enable DTC transfers, add the DTC module under the RSPI HAL module.

The RSPI HAL module supports 8-, 16-, and 32-bit data transfers in both CPU and DTC-based transfer modes. 16- and 32-bit transfers will be endian swapped.

The RSPI HAL drivers also provide supports swapping oftransmit/receive data in byte units for both 16-bit and 32-bit data transfer (big endian to little endian).This feature is applicable only for S5 series MCUs.

The RSPI HAL module supports run time configuration of 8-, 16- and 32-bit data transfers for both CPU and DTC data transfer modes. The DTC transfer size can be any irrespective of user API bit-width.

Performance Notes

The RSPI HAL module can be configured for several different modes, each with different performance characteristics. DTC transfers take slightly longer to setup than CPU-based transfers due to resetting the DTC, but DTC transfers offer greater performance for transfers larger than 1 frame because no intervention is required from the CPU.

Write operations will configure the module for transmit-only mode, disabling the receive interrupt and ignoring incoming data. CPU-based write operations at high bitrates can result in the transmit ISR being constantly called, blocking other code from running. spi_api_t::writeRead and spi_api_t::read operations will configure the module for full duplex mode.

There is a lower limit of 3 SPI clock cycles between transfers resulting in an effective bitrate slower than configured. At high bitrates, the time between transfers can be longer, especially for CPU-based transfers.

The module will wait for the hardware to enter an idle state when in master mode, or all data to be transmitted or received when in slave mode, before invoking the callback.

RSPI HAL Module Limitations

- When the RSPI HAL module is used in slave mode, either the data must be sampled on the even clock edge (that is, CPHA=1) or the master must de-assert the slave select line between the frames when data is sampled on the odd clock edge (that is, CPHA=0). This is a hardware limitation.
- The S124, S128, and S3A6 do not support SSL Level Keep.
- Once the driver has been opened using the DTC or CPU, all transfers must use the bit width configured by the user.
- 16- and 32-bit transfers will be endian swapped.
- The R_RSPI bit rate value must be a positive integer less than 30 MHz or PCLK/2, whichever is smaller.
- r_rspi data transfers will be incomplete when DMAC is used simultaneously by another module. User callback will occur before the data is completely transferred.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.46.4 Including the RSPI HAL Module in an Application

This section describes how to include the SPI HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SPI Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SPI Driver is g_spi0. This name can be changed in the associated Properties window.)

RSPI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_spi0 SPI Driver on r_rspi	Threads	**New Stack> Driver> Connectivity> SPI Driver on r_rspi**

When the SPI Driver on r_rspi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

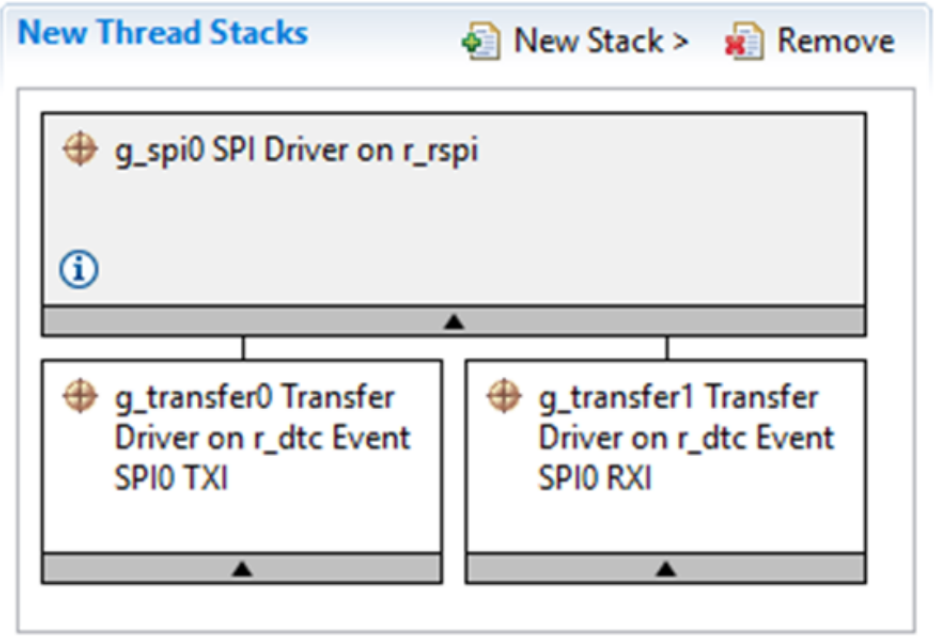


Figure 389: SPI HAL Module Stack

4.2.46.5 Configuring the RSPI HAL Module

The SPI HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the RSPI HAL Module on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.

Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device.
Clock Phase	Data sampling on odd edge, data variation on even edge/Data sampling on even edge, data variation on odd edge Default: Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge. Note: If CPHA is set to 0 in slave mode, slave select should not be fixed at active state and burst transfer should not be performed.
Clock Polarity	Low when idle, High when idle Default: Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	SPI Operation, Clock synchronous operation Default: SPI Operation	Select spi or clock syn mode operation.
SPI Communication Mode	Full Duplex, Transmit Only Default: Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL)	Active Low, Active High Default: Active Low	Select SSL signal polarity.
Select Loopback1	Normal, Inverted Default: Normal	Select loopback1.
Select Loopback2	Normal, Inverted Default: Normal	Select loopback2.
Enable MOSI Idle State	Enable, Disable Default: Disable	Select MOSI idle fixed value and selection.

MOSI Idle State	MOSI Low, MOSI High Default: MOSI Low	Select mosi idle fixed value and selection.
Enable Parity	Enable, Disable Default: Disable	Enable/disable parity.
Parity Mode	Parity Odd, Parity Even Default: Parity Odd	Select parity.
Select SSL Level After Transfer	SSL Level Keep, SSL Level Do Not Keep Default: SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep. Note: If CPHA = 0 in slave mode, slave select level after transfer should not be set to SSL level Keep.
Clock Delay Enable	Clock Delay Enable, Clock Delay Disable Default: Clock Delay Disable	Clock delay enable selection.
Clock Delay Count	Clock Delay 1 thru 8 RSPCK Default: Clock Delay 1 RSPCK	Clock delay count selection.
SSL Negation Delay Enable	Negation Delay Enable, Negation Delay Disable Default: Negation Delay Disable	SSL negation delay enable selection.
Negation Delay Count	Negation Delay 1 thru 8 RSPCK Default: Negation Delay 1 RSPCK	Negation delay count selection.
Next Access Delay Enable	Next Access Delay Enable, Next Access Delay Disable Default: Next Access Delay Disable	Next access delay enable selection.
Next Access Delay Count	Next Access Delay 1 thru 8 RSPCK Default: Next Access Delay 1 RSPCK	Next access delay count selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Receive interrupt priority selection.

Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Error interrupt priority selection.
Byte Swap(Only for S5 series MCUs)Disable	Enable, Disable Default: Disable	Enable byte swapping (applicable only for S5 series MCUs).

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.
RSPI driven slave select currently supports SSL0 only.

Configuration Settings for the SPI HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes Default: 2 Bytes	Transfer size selection.

Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes Default: 2 Bytes	Transfer size selection.

Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

RSPI HAL Module Clock Configuration

The SPI peripheral is clocked via Peripheral Clock A (PCLKA). The clock frequencies are configurable in the ISDE by using the Clocks Tab in the configurator. Invalid selections are indicated in red when selected. Ensure that desired SPI bitrate can be achieved with the stated value of PCLKA. The ISDE will not be indicated if the specified bitrate is not achievable. At run time, the SPI driver will attempt to configure the SPI peripheral to the correct bitrate and will return an error if the desired bitrate cannot be set. The bitrate is calculated via the equations in the table below. If the result of the equation (n) is in the range of 0 to 255, then the bit rate can be achieved.

Baud Rate Calculation Equations

SPI HAL	Bitrate calculation	Description
SPI on SPI	$n = \frac{PCLKA\ (MHz)}{2 * 2^N * B} - 1$	n = Peripheral register value. This has to be in the range of 0 to 255 PLCKA = value of PLCKA in MHz N = 0, 1, 2 or 3 B = Desired Bit Rate

RSPI HAL Module Pin Configuration

The SPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example listing a selection for SPI pins:

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for SPI HAL Module on r_riic

Resource	ISDE Tab	Pin selection Sequence
RSPI	Pins	Select **Peripherals**> **RSPI**> **SPI0_Pin_Option_A/B.**

Note

The selection sequence assumes SPI0 is the desired hardware target for the driver.

Pin Configuration Settings for the SPI HAL Module on r_sci_uart

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, Enabled Default: Disabled	Select Enabled for SPI Operation.
MISO	None, P100, P410 Default: None	MISO Pin selection.
MOSI	None, P101, P411 Default: None	MOSI Pin selection.
RSPCLK	None, P102, P412 Default: None	RSPCLK Pin selection.
SSL0:3	None, P103:106, P413:415 Default: None	SSL0:3 Pin selections.

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.46.6 Using the SPI HAL Module in an Application

The steps in using the SPI HAL module in a typical application are:

Note

The spi_api_t::open API must be called first, but the rest of the calls may be used in any order depending on the application requirements.

1. Initialize the module using the spi_api_t::open API.
2. Write to a slave device by using the spi_api_t::write API.
3. Read from a slave device using the spi_api_t::read API.
4. Close the module by calling the spi_api_t::close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

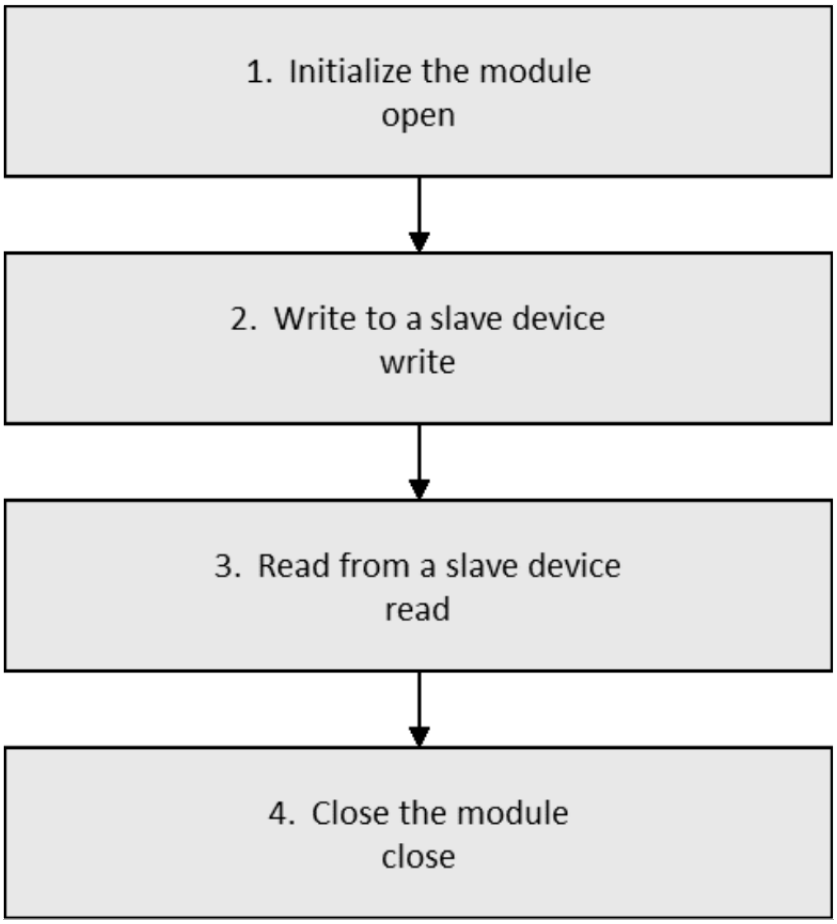


Figure 390: Flow Diagram of a Typical SPI HAL Module Application

4.2.47 UART Driver

4.2.47.1 UART HAL Module Introduction

The UART HAL Module provides a high-level API for industry standard UART serial communications applications and uses the SCI peripherals on the Synergy MCU. A user-defined callback can be created to manage hardware-handshake and data operation, if needed.

UART HAL Module Features

The UART HAL module supports the standard UART protocol. The UART HAL module used in concert with the SCI peripheral in UART mode (UART on SCI) supports the following features (in addition to

the standard UART protocol):

- Full-duplex UART communication
- Simultaneous communication with multiple channels
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code in the argument
- Baud-rate change at run-time
- Hardware resource locking during UART transaction
- CTS/RTS hardware flow control (with an associated IOPORT pin and supported by user-defined callback function)
- Integration with the DTC transfer module
- Abort in-progress read/write operations

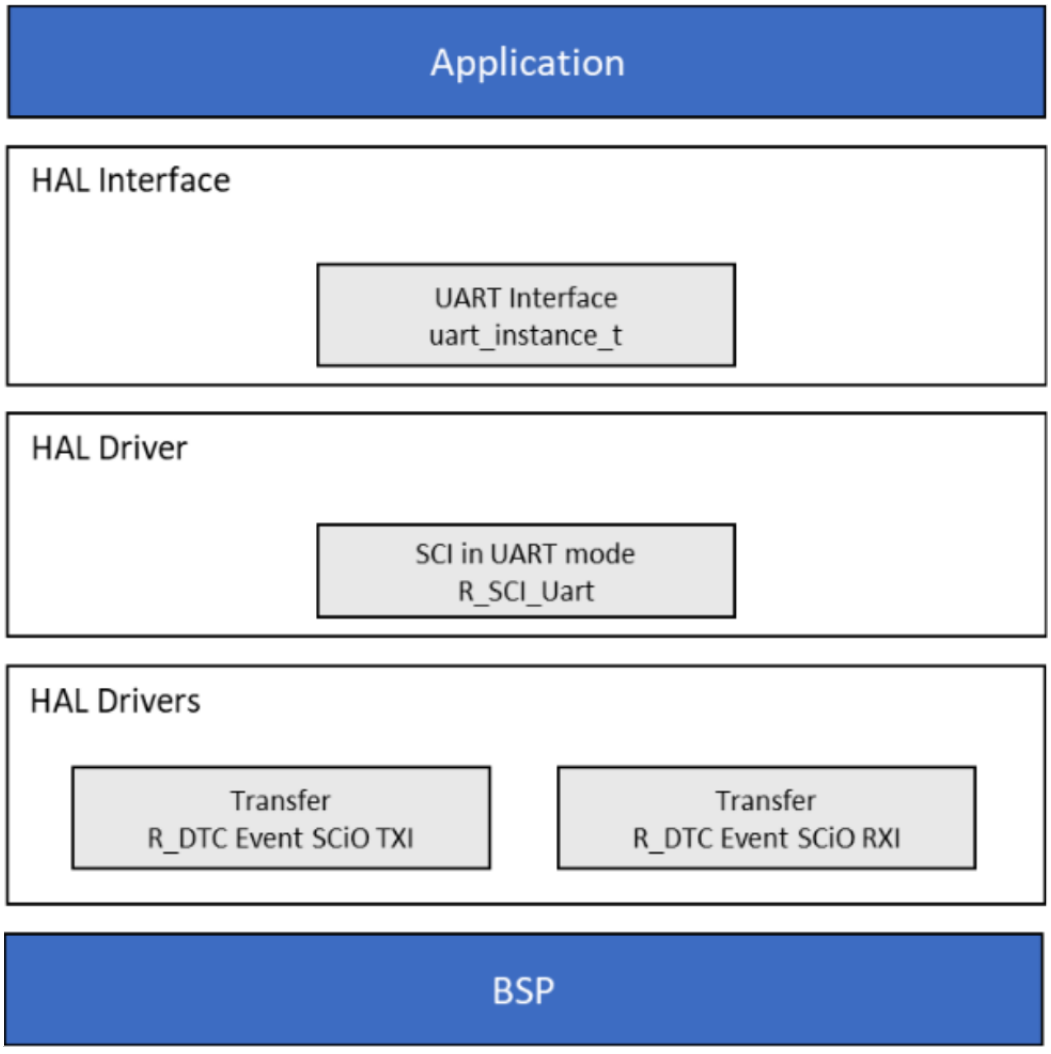


Figure 391: UART HAL Module Block Diagram

UART Hardware Support Details

The following hardware features are, or are not, supported by SSP for the UART (SCI).

Legend:

Symbol	Meaning

✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Serial communication mode: Asynchronous	Serial communication mode: Clock synchronous	Serial communication mode: Smart card
S124	✓	✓	☒
S128	✓	✓	☒
S1JA	✓	✓	☒
S3A1	✓	✓	☒
S3A3	✓	✓	☒
S3A6	✓	✓	☒
S3A7	✓	✓	☒
S5D3	✓	✓	☒
S5D5	✓	✓	☒
S5D9	✓	✓	☒
S7G2	✓	✓	☒

MCU Group	Bit selectable transfer speed	Data Length 7, 8, or 9 bits	Support all Interrupt Sources	Transmission stop bit 1 or 2 bits	Parity: Even parity, odd parity, or no parity	Receive error detection
S124	✓	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Hardware flow control	Transmission and reception	Address match	Address un-match	Start-bit detection	Break detection
S124	✓	Register	☒	☒	✓	☒

S128	✓	Register	☒	☒	✓	☒
S1JA	✓	Register	☒	☒	✓	☒
S3A1	✓	FIFO	☒	☒	✓	☒
S3A3	✓	FIFO	☒	☒	✓	☒
S3A6	✓	FIFO	☒	☒	✓	☒
S3A7	✓	FIFO	☒	☒	✓	☒
S5D3	✓	FIFO	☒	☒	✓	☒
S5D5	✓	FIFO	☒	☒	✓	☒
S5D9	✓	FIFO	☒	☒	✓	☒
S7G2	✓	FIFO	☒	☒	✓	☒

MCU Group	Clock source Internal/ external	Double speed mode	Multi-processor communications	Noise cancellation	Bit rate modulation function	iRDA support
S124	✓	☒	☒	✓	✓	☒
S128	✓	☒	☒	✓	✓	☒
S1JA	✓	☒	☒	✓	✓	☒
S3A1	✓	☒	☒	✓	✓	☒
S3A3	✓	☒	☒	✓	✓	☒
S3A6	✓	☒	☒	✓	✓	☒
S3A7	✓	☒	☒	✓	✓	☒
S5D3	✓	☒	☒	✓	✓	☒
S5D5	✓	☒	☒	✓	✓	☒
S5D9	✓	☒	☒	✓	✓	☒
S7G2	✓	☒	☒	✓	✓	☒

4.2.47.2 UART HAL Module APIs Overview

The UART HAL module interface defines APIs for key features such as opening, closing, reading, writing and setting the baud rate. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

UART HAL Module API Summary

Function Name	Example API Call and Description
open	g_uart0.p_api->open(g_uart0.p_ctrl, g_uart0.p_cfg); Open UART device.

read	g_uart0.p_api->read(g_uart0.p_ctrl, uart0_buf, uart0_rcv_num); Read from UART device. The received bytes are stored directly in the read input buffer, uart0_buf. When a transfer is complete/expected bytes are received, the callback is called with event UART_EVENT_RX_COMPLETE. When the uart_api_t::read API is not called then the bytes will be received in the callback function with event UART_EVENT_RX_CHAR for each byte.
write	g_uart0.p_api->write(g_uart0.p_ctrl, uart0_buf, uart0_send_num) Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE.
baudSet	g_uart0.p_api->baudSet(g_uart0.p_ctrl, (uint32_t)9600); Change baud rate.
infoGet	g_uart0.p_api->infoGet(g_uart0.p_ctrl, &uart_info); Get the driver specific information.
close	g_uart0.p_api->close(g_uart0.p_ctrl); Close UART device.
versionGet	g_uart0.p_api->versionGet(version); Retrieve the API version with the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Channel operates successfully.
SSP_ERR_IN_USE	Control block has already been opened or channel is being used by another instance.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL or configuration structure is NULL.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set.

SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure. Or source/destination address or data size is invalid against data length.
SSP_ERR_NOT_OPEN	The control block has not been opened.
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_RX_ENABLE is set to 0.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.47.3 UART HAL Module Operational Overview

The UART HAL Module manages data flow using the standard UART protocol. The high-level APIs are used to read, write and set the baud rate for the UART interface. In addition, interrupts are typically used to simplify the management of low-level activities.

Note

Interrupts need to be enabled for the following functions to operate successfully.

UART on SCI RXI Interrupt

The RXI interrupt is used to control the flow of data received from the UART port. When the amount of received data reaches the expected read length, the ISR invokes a user-defined callback (sci_uart_instance_ctrl_t::p_callback) with the argument uart_callback_args_t to indicate that the received data is complete. When the External RTS Operation option is enabled, the ISR invokes the UART callback function for the RTS external pin control twice: once at the top of ISR and once at the bottom. You can use the callback function to emulate the RTS function (see the UART on SCI hardware flow-control section); this interrupt is activated as long as reception is enabled in the SCI_UART_CFG_RX_ENABLE configuration parameter.

UART on SCI TXI Interrupt

The TXI interrupt handles consecutive transmissions of data to the UART port as requested by the uart_api_t::write API. When no data is left in the transmit circular buffer, the ISR deactivates the TXI interrupt and activates the TEI interrupt to handle the last sequence in the data transmission. This interrupt is activated in the uart_api_t::write API as long as the transmission is enabled by the SCI_UART_CFG_TX_ENABLE configuration parameter.

UART on SCI TEI Interrupt

The TEI interrupt is used to handle a last data transmission to the UART port requested by uart_api_t::write API; this interrupt is activated by TXI ISR and deactivates itself. The ISR invokes a user-defined callback (sci_uart_instance_ctrl_t::p_callback) with the argument uart_callback_args_t to indicate that the end of data istransmit.

UART on SCI ERI Interrupt

The ERI interrupt is used to handle errors that occur in the UART reception. This interrupt is activated in the uart_api_t::open API as long as the reception is enabled by the SCI_UART_CFG_RX_ENABLE configuration parameter. The ISR invokes a user-defined callback (sci_uart_instance_ctrl_t::p_callback) with the argument uart_callback_args_t to indicate uart_event_t cause of an error.

UART HAL Module Important Operational Notes and Limitations

UART HAL Module Operational Notes

UART on SCI Hardware Flow Control

The SCI hardware module supports hardware flow-control for only one of the RTS or CTS signals at a time. CTS and RTS are multiplexed on the CTSn/RTSn pin so that one of the hardware flow-control signals can be used exclusively depending on the use-case. The UART HAL module expands this specification and allows control of both the CTS and the RTS signal by enabling an additional pin for the RTS signal. To enable this mode, set the UART on SCI configurations as follows:

- Set SCI_UART_CFG_EXTERNAL_RTS_OPERATION to Enable.
- Set uart_cfg_t::ctsrts_en to CTS (true).
- Specify a user-callback function name to "Name of UART callback function for the RTS external pin control" in uart_on_sci_cfg_t::p_extpin_ctrl.

The UART on SCI HAL module invokes the user-callback function from the RXI ISR at the top and at the end of processing.

The callback function argument "level" refers to the signal level on the RTS pin for the selected SCI channel.

Note

The HAL module does not handle the GPIO pin-initialization or control it; instead, the user needs to initialize the GPIO pin before starting the UART reception.

The following figure shows the timing diagram of CTS/RTS hardware flow-control with an external GPIO pin used as the RTS signal:

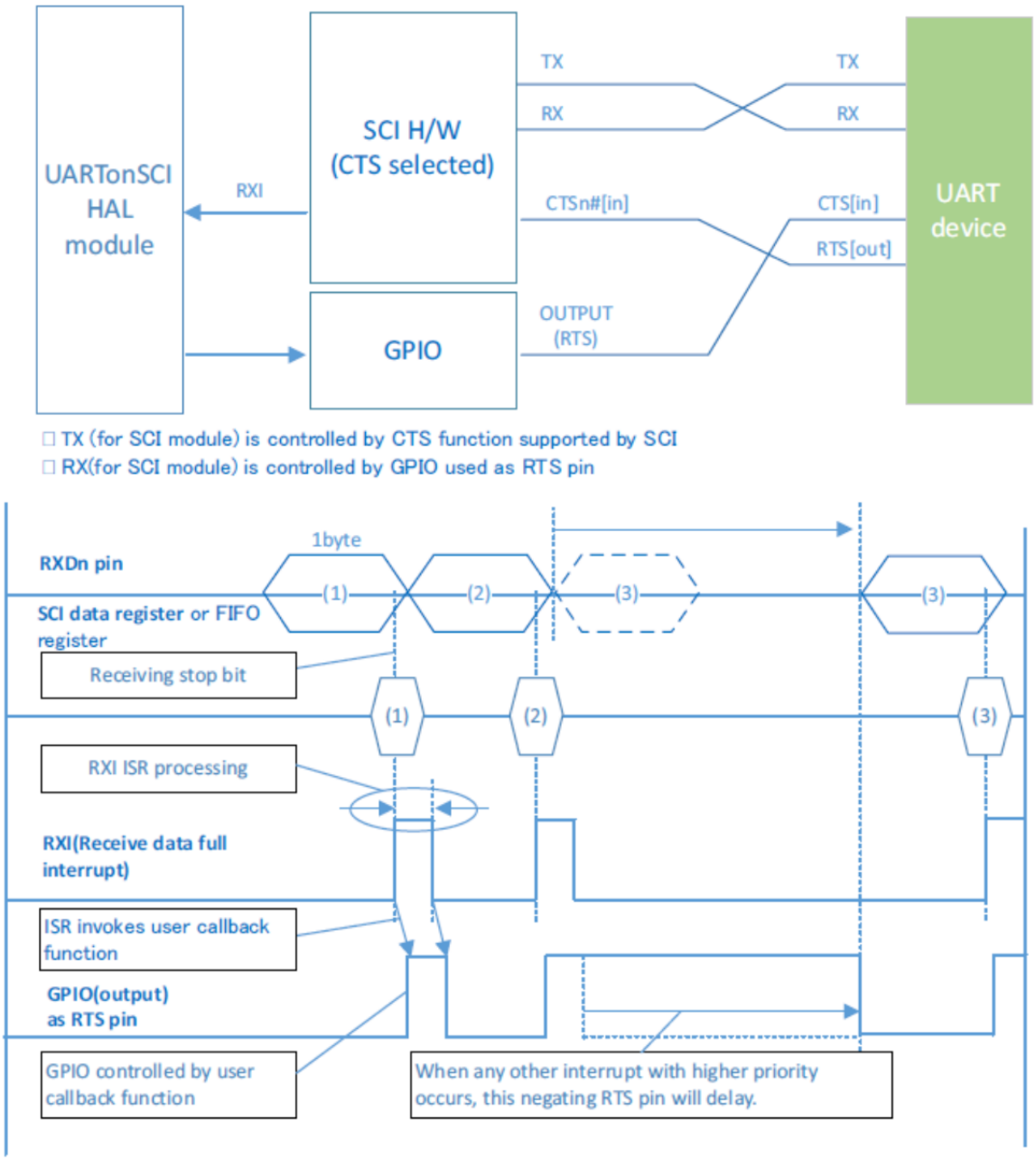


Figure 392: UART HAL Module CTS/RTS Hardware Control with an External GPIO

Note

The UART on SCI module on the SK-S7G2 board uses PORT8 pin0 (pin P800) and J8 to activate the RS232C port on the RS-232C transceiver. Connect pin 1 and pin 2 of J8. Configure pins P800 as IOPORT pins and set its level for the desired operation.

Notes on RS485 Implementations

- For RS485 communication mode, ON pin and RXEN pin should be controlled from the application
- In the case of RS485 Full Duplex communication, configuration of ON pin is not required and in the case of RS485 half-duplex communication, ON should be configured as level Low

- RXEN pin is usually DIP switch (HALF or HD/FD) in Synergy MCUs and is made LOW (ON) for RS485 half-duplex and made HIGH (OFF) for RS485 Full Duplex communication

UART HAL Module Limitations

- The module supports interrupt-based operation but does not support a polled UART mode.
- The module does not support non-buffered UART mode.
- The module does not support Event Link functionality.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read, will arrive via the callback function with event UART_EVENT_RX_CHAR.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.47.4 Including the UART HAL Module in an Application

This section describes how to include the UART HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the UART Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the UART Driver is g_uart0. This name can be changed in the associated Properties window.)

UART HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_uart0 UART on r_sci_uart	Threads	**Threads> Driver> Connectivity> UART Driver on r_sci_uart**

When the UART HAL module on r_sci_uartis added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

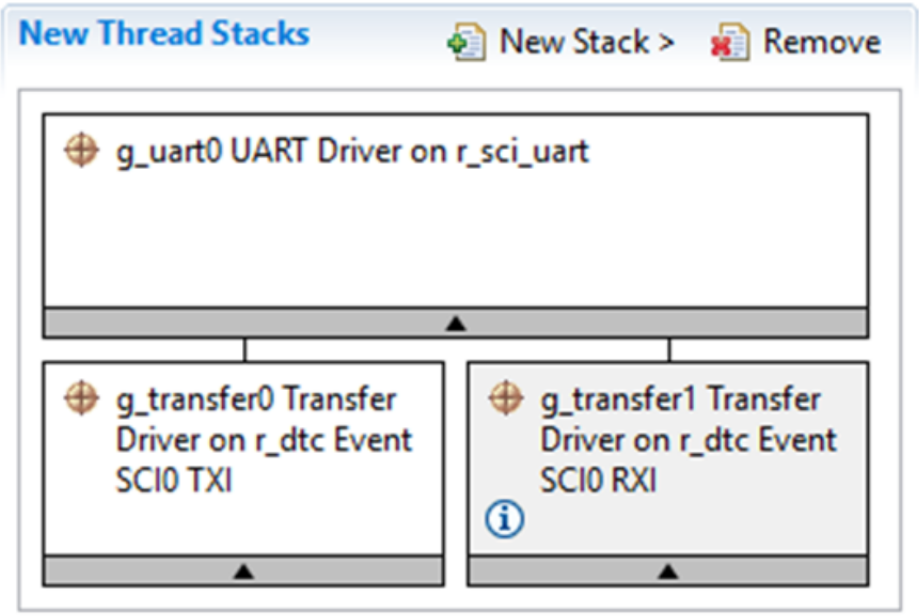


Figure 393: UART HAL Module Stack

4.2.47.5 Configuring the UART HAL Module

The UART HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

ConfigurationSettings for the UART HAL Module on r_sci_uart

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to "Enable" and specify the configuration "Name of UART callback function for the RTS external pin control".

Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to "Disable" reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting "Disable" to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set "Disable" to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select "CTS" for this configuration parameter and enable the configuration "External RTS Operation" specifying the configuration "Name of UART callback function for the RTS external pin control".
Name of UART callback function to be defined by user	user_uart_callback	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set "Falling Edge" to this configuration.
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection.

Receive FIFO Trigger Level	One, Max Default: Max	Receive FIFO trigger level selection: One: an interrupt occurs for every byte received. Max: an interrupt will be triggered if either of the below conditions are met: a) The FIFO is filled to the Max level (15). b) 15bit times have occurred with no data received.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Error interrupt priority selection.
Baud rate Percent Error	Value must be greater than 0.0 and less than 15.0 Default; 2.0	Maximum baudrate percent error allowed in order for the module to function.
UART Communication Mode	RS232, RS485 Default: RS232	UART communication mode selection, usually it is RS232 mode.
UART RS485 Communication Mode	Half Duplex, Full Duplex Default: Half duplex	UART RS485 communication mode selection as half duplex or full duplex.
RS485 DE Port	00 to 11 Default: 09	Select the port number of Driver Enable Pin.
RS485 DE Pin	00 to 15 Default: 14	Select the pin number of Driver Enable Pin.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the UART HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	True, False Default: True	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection.
Name	g_transfer1	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

When using the UART with CTS and RTS function simultaneously, the transfer driver cannot be used; please delete all transfer drivers on the low level. After being deleted, the optional transfer driver will display in pink, meaning the driver is recommended but optional.

UART HAL Module Clock Configuration

The SCI UART peripheral uses PCLKA as its clock source (PCLKB for S124) or an external clock from the SCKn pin for the selected channel n.

UART HAL Module Pin Configuration

The SCI UART peripheral uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the UART pins.

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for UART HAL Module on SCI

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals> Connectivity: SCI > SCI0**

Pin Configuration Settings for the UART HAL Module on r_sci_uart

Pin Configuration Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only (Default: Mixed)	Pin grouping selection.
Operation Mode	Disabled, Custom, Asynchronous UART, Simple SPI, Simple I2C, Synchronous UART, SmartCard (Default: Simple SPI)	Select Operation Mode for UART on SCI.
TXD_MOSI	None, P411, P101 (Default: P411)	TXD Pin.
RXD_MISO	None, P410, P100 (Default: P410)	RXD Pin.

SCK	None, P412, P102 (Default: P412)	SCK Pin.
CTS_RTS_SS	None, P413, P103 (Default: None)	CTS Pin.
SDA	Disabled	SDA Pin (when Simple I2C is used).
SCL	Disabled	SCL Pin (when Simple I2C is used).

Note

The example settings are for a project using the Synergy S7G2 MCU Group and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.47.6 Using the UART HAL Module in an Application

The steps in using the UART HAL module in a typical application are:

1. Initialize the UART HAL Module using the uart_api_t::open API.
2. Set Baud Rate with the uart_api_t::baudSet API (if needed.)
3. Read and Write data as needed using the uart_api_t::read and uart_api_t::write APIs and callbacks.
4. Read or Write operations can be aborted using uart_api_t::communicationAbort API if required.
5. Close the UART HAL module using the uart_api_t::close API as needed.

These common steps are illustrated in a typical operational flow diagram in the following figure:

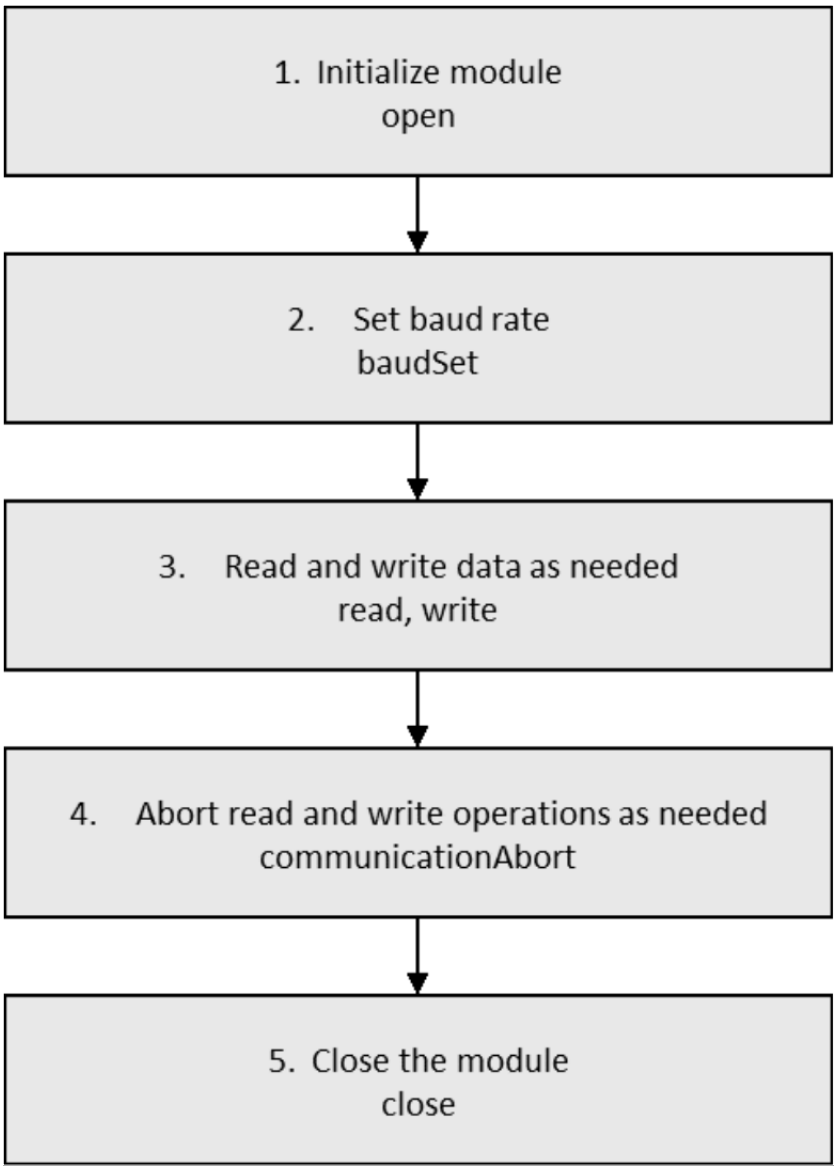


Figure 394: Flow Diagram of a Typical UART HAL Module Application

4.2.48 Watchdog Driver

4.2.48.1 Watchdog Timer HAL Module Introduction

The WDT (Watchdog Timer) HAL module provides a high-level API for critical timing applications and uses the WDT peripheral on the Synergy MCU. A user-defined callback can be created to respond to event notifications.

Watchdog Timer HAL Module Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
    - Resetting of the device
    - Generation of an NMI
- Supports the Watchdog Timer (WDT) peripheral, which uses an external clock.
- The WDT can be configured in register start mode through the WDT registers.
- Supports automatic hardware configuration after reset.
- The WDT can be started from the application.

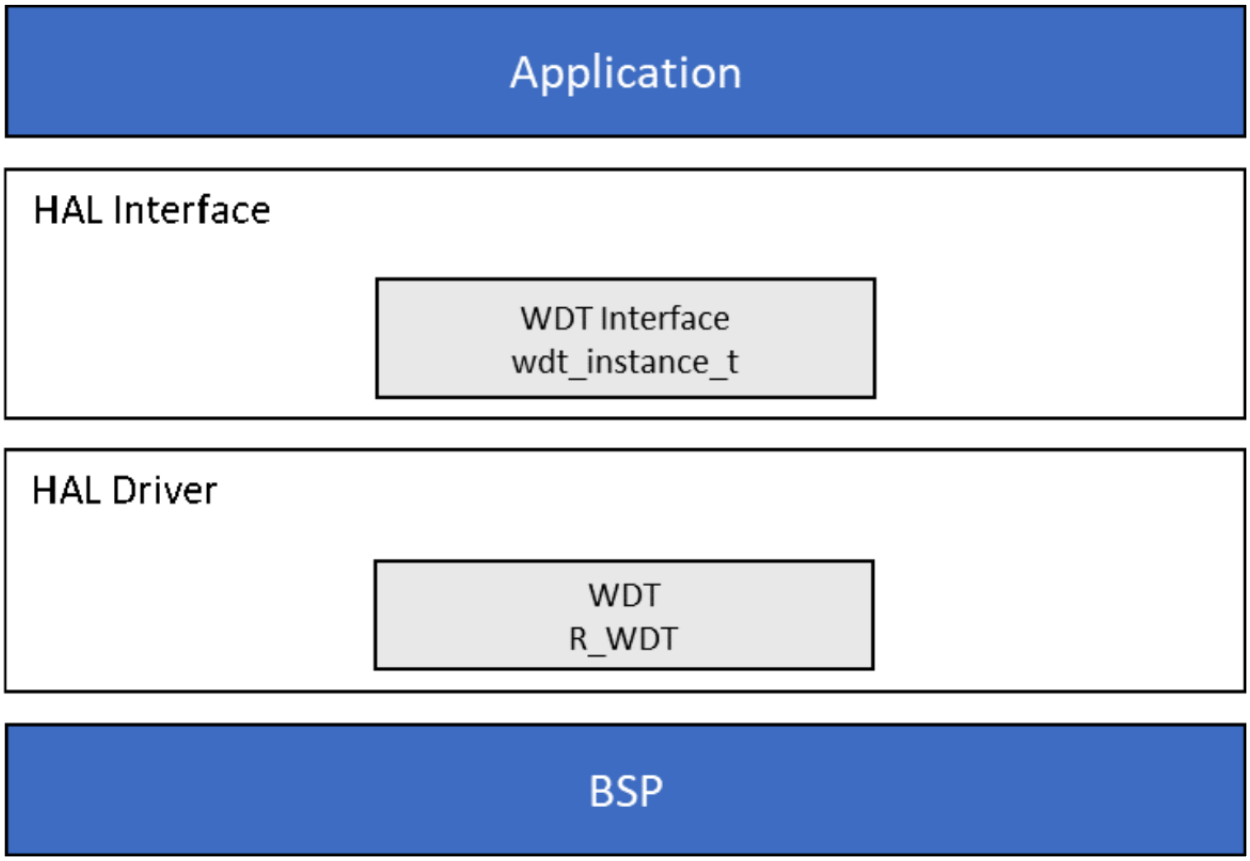


Figure 395: Watchdog Timer HAL Module Block Diagram

Watchdog Timer Hardware Support Details

The following hardware features are, or are not, supported by SSP for WDT:

Legend:

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

MCU Group	Clock Divide by 4, 64, 128, 512, 2,048, or 8,192	Count down	Register start mode	Auto start mode	Reset output	Interrupt request output
S124	✓	✓	✓	✓	✓	✓
S128	✓	✓	✓	✓	✓	✓
S1JA	✓	✓	✓	✓	✓	✓
S3A1	✓	✓	✓	✓	✓	✓
S3A3	✓	✓	✓	✓	✓	✓
S3A6	✓	✓	✓	✓	✓	✓
S3A7	✓	✓	✓	✓	✓	✓
S5D3	✓	✓	✓	✓	✓	✓
S5D5	✓	✓	✓	✓	✓	✓
S5D9	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓

MCU Group	Sleep mode count stop control output	Event link function through ELC HAL driver	Window function	Conditions for stopping the Counter – reset/under flow refresh error	Refresh error and under flow error detect	Reading the counter value
S124	☒	☒	✓	✓	✓	✓
S128	☒	☒	✓	✓	✓	✓
S1JA	☒	☒	✓	✓	✓	✓
S3A1	☒	☒	✓	✓	✓	✓
S3A3	☒	☒	✓	✓	✓	✓
S3A6	☒	☒	✓	✓	✓	✓
S3A7	☒	☒	✓	✓	✓	✓
S5D3	☒	☒	✓	✓	✓	✓
S5D5	☒	☒	✓	✓	✓	✓
S5D9	☒	☒	✓	✓	✓	✓
S7G2	☒	☒	✓	✓	✓	✓

4.2.48.2 Watchdog Timer HAL Module APIs Overview

The WDT HAL module defines APIs for opening, refreshing, reading and getting status. A complete list of the available APIs, an example API call and a short description of each can be found in the

following table. A table of status return values follows the API summary table.

Watchdog Timer HAL Module API Summary

Function Name	Example API Call and Description
cfgGet	g_wdt0.p_api->cfgGet(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
open	g_wdt0.p_api->open(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
refresh	g_wdt0.p_api->refresh(g_wdt0.p_ctrl); Refresh the watchdog timer.
statusGet	g_wdt0.p_api->statusGet(g_wdt0.p_ctrl, &status); Read the status of the WDT.
statusClear	g_wdt0.p_api->statusClear(g_wdt0.p_ctrl, clear); Clear the status flags of the WDT.
counterGet	g_wdt0.p_api->counterGet(g_wdt0.p_ctrl, &counter); Read the current WDT counter value.
timeoutGet	g_wdt0.p_api->timeoutGet(g_wdt0.p_ctrl, &timeout); Read the watchdog timeout values.
versionGet	g_wdt0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully executed.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.

SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.48.3 Watchdog Timer HAL Module Operational Overview

Synergy MCUs have two watchdog peripherals- the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

- The WDT can be started from the application.
- The WDT can be configured in register start mode through the WDT registers. The WDT can also be configured by hardware automatically after reset using parameters stored in Option Function Select Register 0 (OFS0).
- The IWDT has its own clock source which improves safety.
- The IWDT is configured by hardware automatically after reset using parameters stored in the Option Function Select Register 0 (OFS0).

Watchdog Timer HAL Module Important Operational Notes and Limitations

Watchdog Timer HAL Module Operational Notes

The WDT HAL module configures the WDT Interface. When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:

- Resetting of the device
- Generation of an NMI

The following figure shows an example of the operation of the WDT. When refreshed in the valid refresh period of the counter the timer count value is reset. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an NMI.

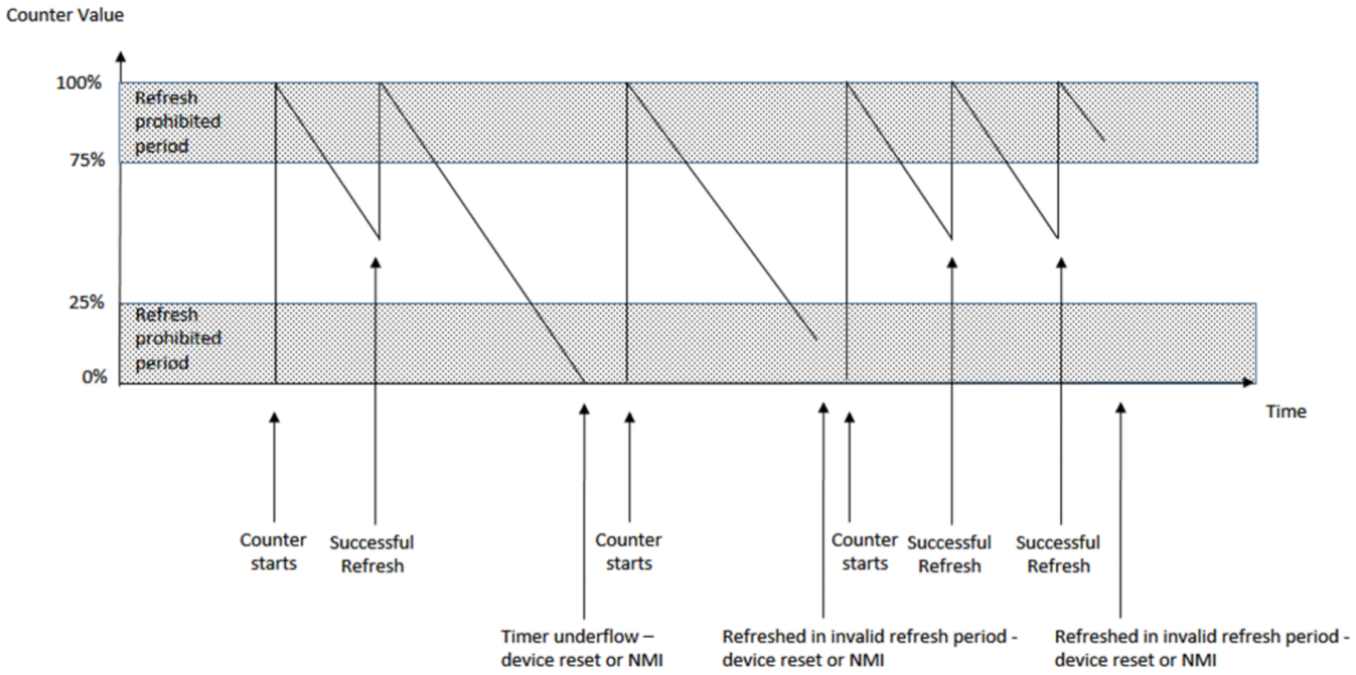


Figure 396: Watchdog Timer HAL Module Operational Diagram

The WDT can be configured in register start mode through the WDT registers. The WDT can also be configured by hardware automatically after reset using parameters stored in Option Function Select Register 0 (OFS0) as displayed in the following table.

All series of Synergy microcontrollers have an option-setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

The following table details which parameters of the IWDT can be configured by the OFS registers.

Note

The IWDT can only be configured via the OFS registers. The IWDT does not support Register Start mode.

Control	Description
IWDT Start Mode Select	Automatically starts the IWDT after a Reset, if enabled.
IWDT Timeout Period	Specifies the IWDT timeout (number of clock cycles) 128 cycles 512 cycles 1024 cycles 2048 cycles
IWDT-Dedicated Clock Frequency Division Ratio	1 1/16 1/32 1/64 1/128 1/256

IWDT Window End Position	25% 50% 75% 100% (no window end position set)
IWDT Window Start Position	25% 50% 75% 100% (no window start position set)
IWDT Reset Interrupt Request	The IWDT can either generate an Interrupt Signal or a Reset signal.
IWDT Stop Control	The IWDT can continue to count or Stop counting in Low Power Mode.

Note

For further information on the contents of the OFS0 register, see the Synergy MCU hardware manual.

The OFS register values are set via the properties dialog of the **BSP** tab of Synergy Configuration editor as shown in the figures below.

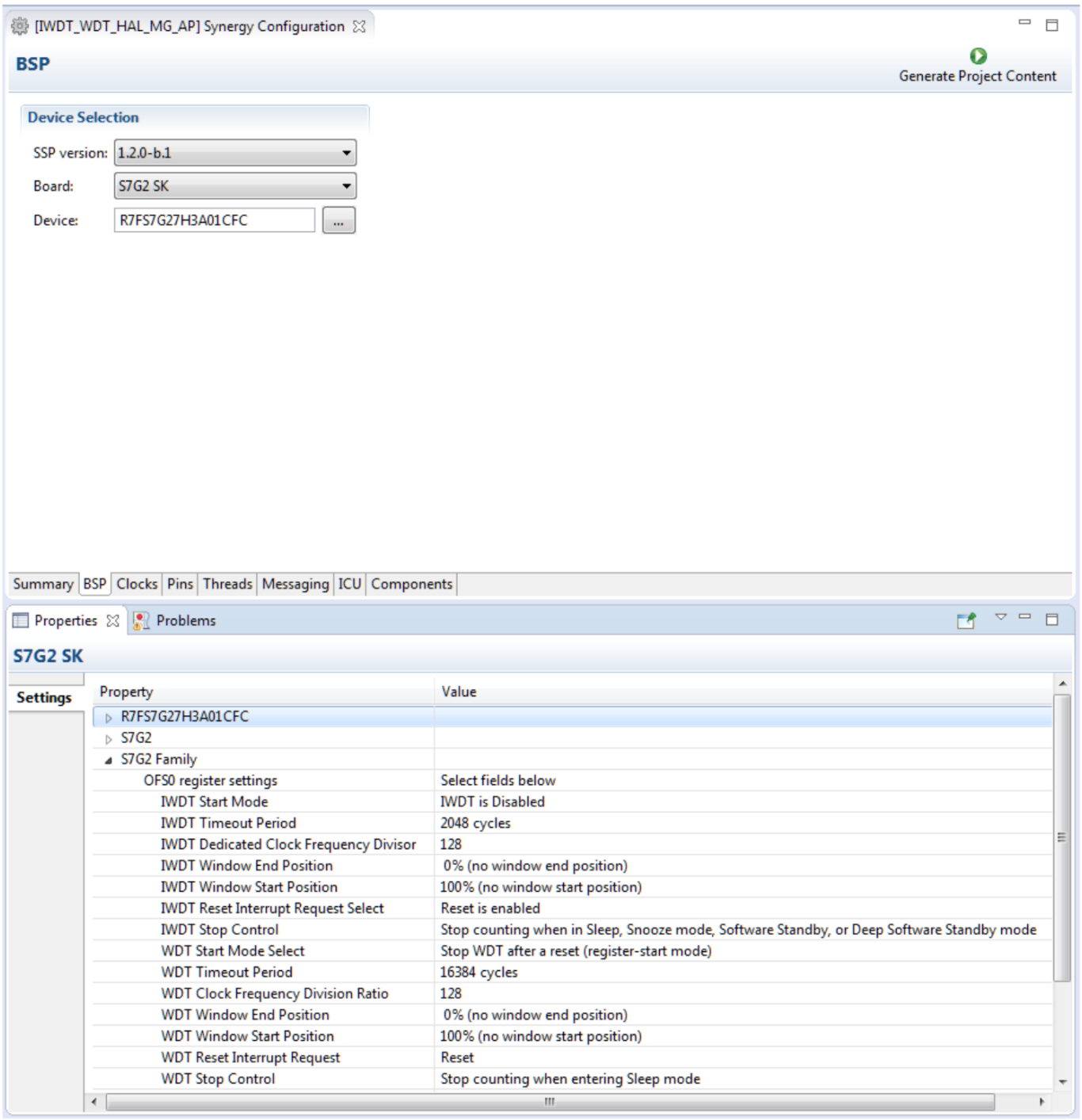


Figure 397: Watchdog Timer HAL Module Configuration Screens

WDT HAL Module Period Calculation

The WDT operates from PCLKB. Assuming largest parameters for the WDT and a PCLKB of 60 MHz, the time from the last refresh to device reset or NMI generation will be just over 2.2 seconds as detailed below.

PLCKB = 60 MHz

Clock division ratio = PCLKB/8192

Timeout period = 16384 cycles

WDT clock frequency = 60 MHz / 8192 = 7.324 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 cycles = 2.23 seconds

Triggering DMAC/DTC with the WDT HAL Module

To trigger a transfer of data using the DMAC or DTC peripheral when the WDT counter underflows or when a refresh is attempted outside of the valid refresh period, configure the WDT to generate an NMI and configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_WDT_UNDERFLOW. See the associated User Guide (DMAC, DTC) for further information.

Triggering Event Link Controller Events with the WDT HAL Module

The WDT can trigger the start of another peripheral using the Event Link Controller (ELC). Refer to the ELC User Guide for a complete list of available peripherals.

Watchdog Timer HAL Module Limitations

- When using a J-Link debugger the WDT counter does not count and therefore will not reset the device or generate an NMI.
- When there is no active task ready to run, ThreadX puts the MCU into sleep mode. If the WDT is configured to stop the counter in low power mode, then your application must restart the timer when used with the ThreadX RTOS.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.48.4 Including the Watchdog Timer HAL Module in an Application

This section describes how to include the Watchdog Timer HAL Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Watchdog Timer Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Watchdog Timer Driver is g_wdt0. This name can be changed in the associated Properties window.)

Watchdog Timer HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_wdt0 Watchdog Driver on r_wdt	Threads	**New Stack>Driver> Monitoring> Watchdog Driver on r_wdt**

When the Watchdog Timer Driver on r_wdt is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band

are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

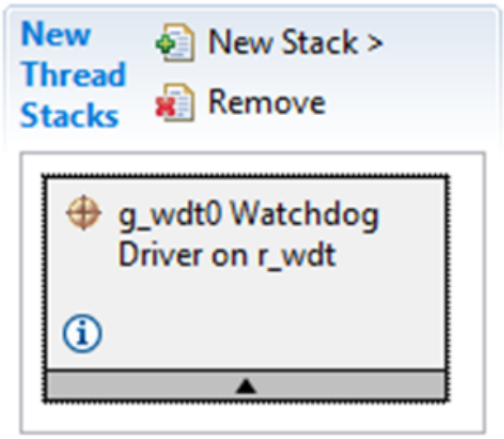


Figure 398: Watchdog Timer HAL Module Stack

4.2.48.5 Configuring the Watchdog Timer HAL Module

The Watchdog Timer HAL Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Watchdog Timer HAL Module on r_wdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_wdt0	Module name.
Start Mode	Register, Auto Default: Register	Configures the start mode as register start or auto-start.

Start Watchdog After Configuration	True, False Default: True	Controls whether WDT is started during initialization.
Timeout	1024 cycles, 4096 cycles, 8192 cycles, 16384 cycles Default: 16384 cycles	WDT timeout period.
Clock Division Ratio	PCLK/4, PCLK/64, PCLK/128, PCLK/512, PCLK/2048, PCLK/8192 Default: PCLK/8192	WDT clock divider.
Window Start Position	100% (Window Position Not Specified), 75%, 50%, 25% Default: 100% (Window Position Not Specified)	Permitted refresh period start postion.
Window End Position	0% (Window Position Not Specified), 25%, 50%, 75% Default: 0% (Window Position Not Specified)	Permitted refresh period end postion.
Reset Control	Reset Outpout, NMI Generated Default: Reset Output	Select whether WDT should reset the MCU or generate an NMI.
Stop Control	WDT Count Enabled in Low Power Mode, WDT Count Disabled in Low Power Mode Default: WDT Count Disabled in Low Power Mode	Select whether the WDT should stop counting in low power modes.
NMI Callback	NULL	Callback. A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configure Option Function Select Register 0 (OFS0) for the WDT HAL Module

All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO. See the description in the operational overview section earlier in this document.

Configuring the Interrupts for the WDT HAL Module

Configure the WDT interrupts in the same way as configuring the other options for the WDT module. If the WDT is configured to generate an NMI interrupt on underflow or invalid refresh, the interrupt must be enabled in the BSP.

To enable interrupts, set the priority of the **CWDT**> **CWDT NMIUNDF N n**. This sets BSP_IRQ_CFG_WDT_UNDERFLOW in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the CWDT NMIUNDF N interrupt is enabled in the BSP, the corresponding ISR will be defined. The ISR will call a user callback function if one was registered in the wdt_api_t::open API.

Watchdog Timer HAL Module Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the clock configurator in the ISDE or using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

Watchdog Timer HAL Module Pin Configuration

The WDT does not require pins for its operation.

4.2.48.6 Using the Watchdog Timer HAL Module in an Application

The typical steps in using the Watchdog Timer HAL module in an application are:

1. Initialize the WDT HAL module in either register start mode or auto-start mode using the wdt_api_t::open API.
2. Read the configuration of the WDT HAL module in either register start mode or auto start mode with the wdt_api_t::cfgGet API.
3. Refresh the watchdog timer using the wdt_api_t::refresh API.
4. Read the WDT status flags using the wdt_api_t::statusGet API.
5. Clear the status flags and error flags of the WDT HAL module using the wdt_api_t::statusClear API.
6. Read the current WDT counter value using the wdt_api_t::counterGet API.
7. Read the watchdog timeout values using the wdt_api_t::timeoutGet API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

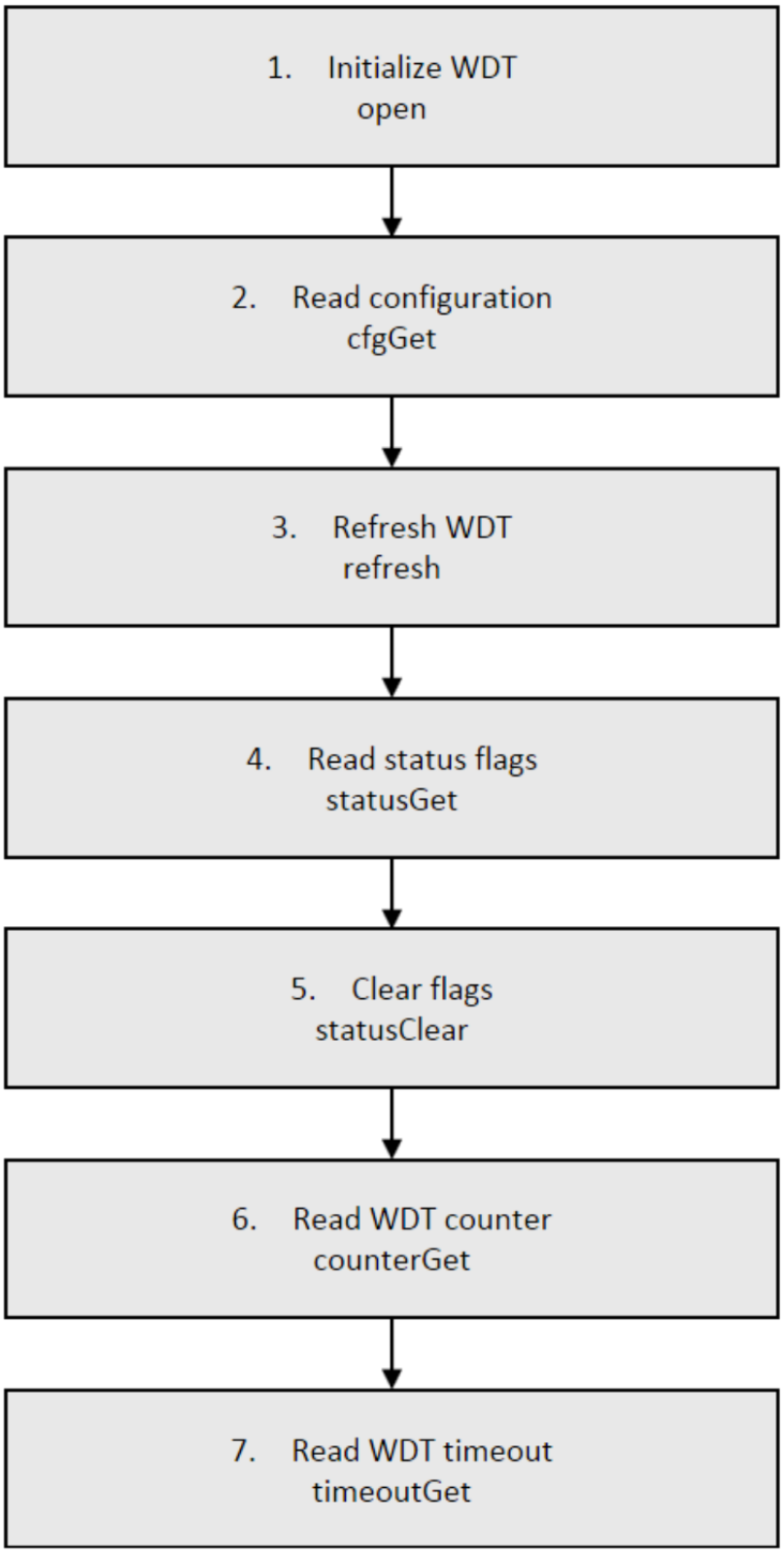


Figure 399: Flow Diagram of a Typical Watchdog Timer HAL Module Application

4.3 Azure RTOS Modules

The following documents for Renesas SSP Azure RTOS modules cover a wide range of topics and are organized to make it easy to find the specific information needed to accelerate your development process. Follow the below pointers to more quickly find the information you are looking for:

- Azure RTOS components are organized by key application functions: ThreadX RTOS support, FileX file system support, GUIX graphics user interface support, USBX Universal Serial Bus interface support, NetX and NetX Duo networking protocol support. The below articles are organized using these same key applications areas.
- Some Azure RTOS components have an Overview document that describes many of the key elements and concepts. The Overview document can also include common information that would be redundant to include in all the separate Module Overviews.
- Some Azure RTOS components have a Source module document that describes the configurable options available within the source module. These documents, and the associated Azure RTOS User's Manuals, (available in the X-Ware Component Documents package at the bottom of this web page: https://www.renesas.com/synergy/ssp) should be understood before attempting to modify any source configuration settings).
- Some Azure RTOS components have a Port Framework document that describes the details of the interface between SSP and an Azure RTOS Component. For example, the NetX Port Ethernet module (sf_el_nx) interfaces the NetX and Net Duo software with the Synergy hardware.
- Some Azure RTOS components have a variety of protocols or similar lower level functions. These are typically included in SSP as separate modules and each has a Module Overview document that explains their operation. For example, there are separate Module Overviews for DHCP, FTP and Telnet.
- Some Azure RTOS networking components have a Module Overview that covers more than one protocol. Often this is because the NetX and NetX Duo implementations are very similar and any differences can be easily identified in the combined document- thus eliminating unneeded duplication. (NetX Duo supports both IPv4 and IPv6- while NetX supports only IPv4. Operational differences can be easily explained where needed.) For example, the NetX/NetX Duo HTTP Server, NetX/NetX Duo HTTP Client Module Overviews cover both the NetX and NetX Duo implementations of these standard functions.

ThreadX

ThreadX Overview

FileX

FileX on Block Media

FileX Source

GUIX

GUIX Port

GUIX Source

LevelX

LevelX Port Framework on sf_el_lx_nor

NetX/NetX Duo

NetX Port Ether

NetX Port Using PPP

NetX/NetX Duo Source

Azure RTOS NetX Overview

Azure RTOS NetX Duo Overview

NetX/NetX Duo Auto IP

NetX/NetX Duo BSD Support

NetX/NetX Duo DHCP Client

NetX/NetX Duo DHCP Server

NetX Duo DHCPv6 Client

NetX Duo DHCPv6 Server

NetX/NetX Duo DNS Client

NetX/NetX Duo FTP Client

NetX/NetX Duo FTP Server

NetX/NetX Duo HTTP Client

NetX/NetX Duo HTTP Server

NetX Duo HTTP Client (HTTPS/HTTPS 1.1)

NetX/NetX Duo HTTP/HTTPS Web Server Framework

NetX/NetX Duo SMTP Client

NetX/NetX Duo SNMP Agent

NetX/NetX Duo SNTP Client

NetX/NetX Duo POP3 Client

NetX/NetX Duo Telnet Client

NetX/NetX Duo Telnet Server

NetX/NetX Duo TFTP Client

NetX/NetX Duo TFTP Server

NetX Duo MQTT Client

NetX Duo NAT

NetX Duo TLS Session

NetX Duo DTLS Session

NetX Duo mDNS/DNS-SD

USBX

Azure RTOS USBX Overview

USBX Source

USBX Port

USBX Device Class CDC-ACM

USBX Device Class HID

USBX Device Class Mass Storage

USBX Host Class CDC-ACM

USBX Host Class HID

USBX Host Class HUB

USBX Host Class Printer USBX Host Class Mass Storage

USBX Host Class Video

4.3.1 ThreadX Overview

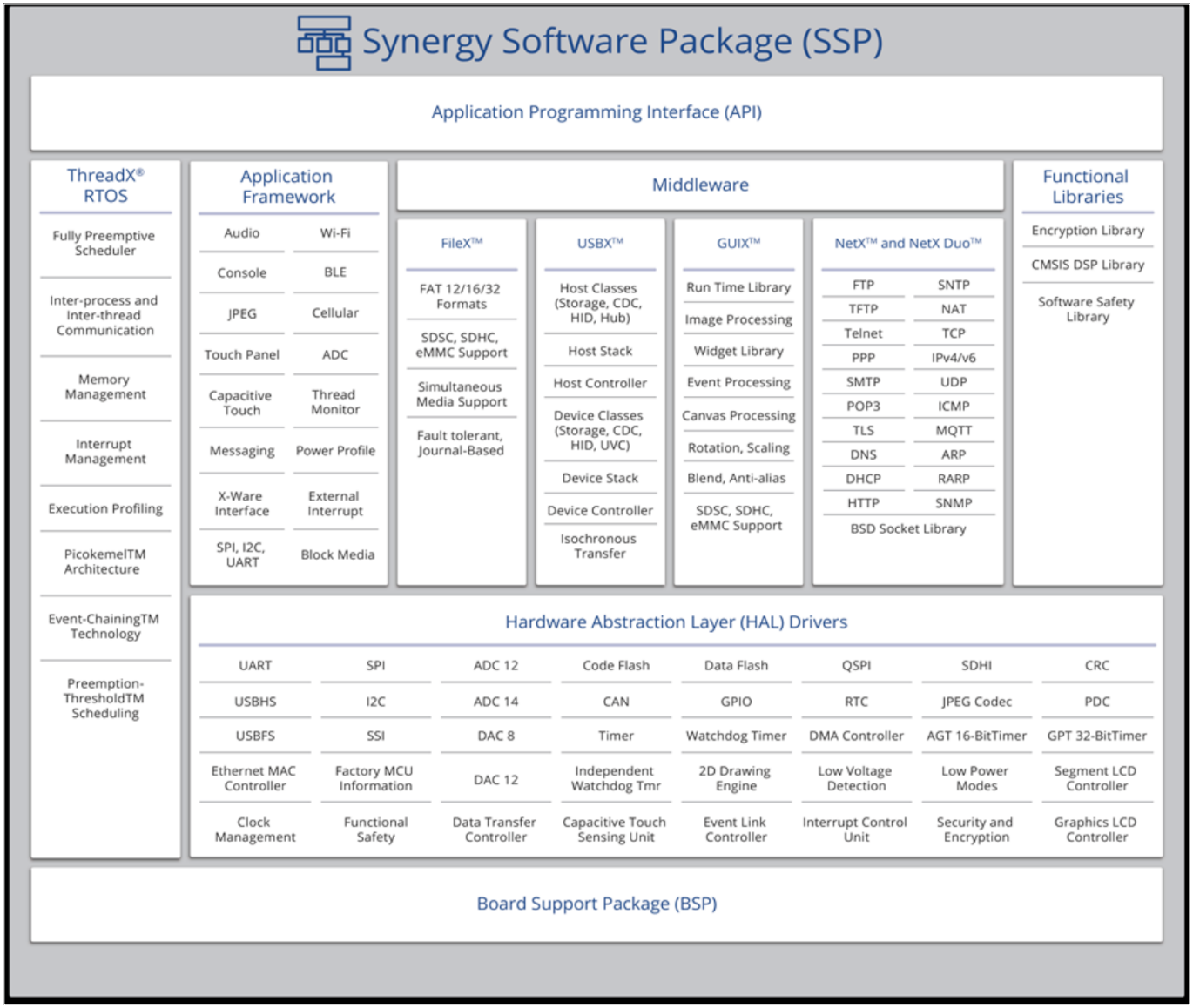
4.3.1.1 Azure RTOS ThreadX Module Introduction

The Azure RTOSThreadX kernel (tx) is integrated into the SSP. The Azure RTOS ThreadX kernel is the foundation of multi-threaded SSP applications. It provides thread creation and synchronization services, including message queues, counting semaphores, mutexes, event flags, memory block pools, memory byte pools, and application timers.

Azure RTOS ThreadX Module Features

- Implements the ThreadX RTOS in the Synergy Platform
- Automatically added when a thread is created
- Source code can be added using the ThreadX source module
- Provides threaded services for:

    - Synchronization
    - Messaging queues
    - Counting semaphores
    - Mutexes
    - Event flags
    - Memory block and byte pools
- Application timers



4.3.1.2 Azure RTOS ThreadX Module Operational Overview

The Azure RTOS ThreadX on tx module implements the ThreadX RTOS kernel within the Synergy Software Package. It provides all the scheduling, inter-process communications, memory management, interrupt management and a variety of other RTOS related functions. Typically, application code simply makes use of the functions provided by the kernel. The following operational notes outline some of the areas where the application may benefit from a more detailed interaction with ThreadX and ThreadX source.

Azure RTOS ThreadX Module Important Operational Notes and Limitations

Azure RTOS ThreadX Module Operational Notes

Preemption Threshold

The preemption threshold feature is disabled in the provided ThreadX library to reduce code size. If preemption support is needed, then ThreadX source should be included in the project and **Preemption Threshold** should be set to **Enabled** in the ThreadX source properties. The preemption threshold for a thread is initially set by the configurator to the thread priority. It can be changed by the tx_thread_preemption_change function.

IAR Library Support (IAR Only)

Thread-safe support for the IAR tools is easily enabled by making the following setting in the **Properties** of the ThreadX Source Module. In the table, ISDE Property refers to the **Properties** tab name in the e^2 studio ISDE.

ThreadX Source IAR Library Support Configuration

ISDE Property	Setting	Description
IAR Library Support	Enabled, Disabled (Default: Disabled)	If enabled, defines TX_ENABLE_IAR_LIBRARY_SUPPORT to provide thread safe access to IAR standard library functions such as malloc() and printf().

Also, add the following line to the linker control file (if not already in place):

@ code

initialize by copy with packing = none { section __DLIB_PERTHREAD };

// Required in a multi-threaded application.

@endcode

Note

*Make sure the ThreadX Source module is added to the **Threads** tab (HAL/Common) before modifying TX_ENABLE_IAR_LIBRARY_SUPPORT.*
*The TX_ENABLE_IAR_LIBRARY_SUPPORT macro enables thread creation and destruction extensions which are required for thread safe access to the IAR runtime library. The TLS (Thread Local Storage) memory is allocated from the heap for each instance. Currently, SSP **does not** implement the synchronization mechanisms that are required to use the library functions in a thread safe manner. Using the library with just the features provided is safe only if the library functions are accessed in a non-reentrant manner, that is, only a single thread makes calls to this library. To get full multithreaded support, refer to the IAR Dlib Thread Support document from IAR for more information on how to implement the synchronization mechanisms. The implementation will need to be done in the tx_src_user.h file which is an SSP generated file.*

ThreadX Source Advanced Configurations

If the ThreadX Source module is added to the project, the Properties window provides advanced configurations for the ThreadX source library. Highlight a configuration option to view a description of the option in the bottom left corner of the e^2 studio GUI. If the configuration option entry field is

empty, the default value will be used. Refer to the Configuration Options chapter of the *ThreadX User Guide* for more information.

At the end of the advanced properties, there are TX_<COMPONENT>_EXTENSION macros, where <COMPONENT> identifies the type of extension. These extension macros are for advanced use cases only. In most projects, they are not modified. Some cases, such as BSD support for NetX, require use of extension macros and explicitly describe how they should be used.

Azure RTOS ThreadX Module Limitations

- Hardware stack monitoring is not supported for the ARM CM23 and CM0+ MCUs. If the ThreadX Source is used with these MCUs, use the SSP configurator to set the Enable Hardware Thread Stack Monitoring property to Disabled.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.1.3 Using the Azure RTOS ThreadX Module in an Application

During the configuration of the project, ThreadX is added automatically when a new Thread is created.

ThreadX objects, including mutexes, semaphores, event flags and queues can be added to Threads in the <Thread Name> Objects section, where <Thread Name> represents the name of an application thread. For example, to add a new queue to a thread, highlight the thread and select New Object > Queue.

To access the queue from a different thread, include the header file of the thread where the queue is defined. For example, if an application has a thread named main_thread with a queue named message_queue, a file named main_thread.h will be generated with an extern for message_queue. If the application also has a thread named background_thread, and the background_thread requires access to the message_queue, main_thread.h should be included in background_thread_entry.c to ensure background_thread_entry.c has access to the definition of message_queue.

Threadx Source configurator property **Memory section for Trace Buffer** relocates the trace buffer to the memory section indicated by the property. This requires enabling the property **Event Trace**. By default the trace buffer will be placed in the .bss section of memory

To relocate the trace buffer:

1. Enable the **Event Trace** property
2. Input the memory section where you want to put trace buffer

Note

It is the developer's responsibility not to relocate the trace buffer to the restricted memory sections. Developers are responsible for inputting the appropriate memory section if they want to relocate the trace buffer.

If the selected device is from a family of MCUs that have less memory, please resize the trace buffer from default size = 65536 to a lesser size so as to avoid RAM overflows by using configurator property **Trace Buffer Size** before enabling the **Event Trace**.

To restart a user defined thread created by Synergy

Applications trying to restart a thread after tx_thread_terminate/tx_thread_reset/tx_thread_resume call have to use tx_semaphore_put (&g_ssp_common_initialized_semaphore) before restarting.

4.3.2 FileX on Block Media

4.3.2.1 FileX On Block Media Framework Module Introduction

The FileX Port on Media Framework module supports the Azure RTOS FileX system, a complete FAT and exFAT format media and a file management system for deeply embedded applications. FileX is highly optimized for both size and performance.

FileX On Block Media Framework Module Features

Single Partition (Uses total available memory size):

- Supports FAT32, FAT16, FAT12 and exFAT filesystems
- Supports single FAT and exFAT partition operations in all SSP block media, that is, sf_block_media_lx_nor, sf_block_media_sdmmc, sf_block_media_qspi and sf_block_media_ram.
- Multiple FileX objects (that is, media, directories, and files, only limited by available memory)
- Dynamic FileX object creation/deletion
- Flexible memory usage
- Size scales automatically
- Small footprint
- Complete integration with ThreadX

Multi Partition:

- Supports FAT32, FAT16 and exFAT filesystem.
- Supports the creation of any number of partition on single physical media, and supports all other fileX operations on them, for underlying block media sf_block_media_lx_nor and sf_block_media_sdmmc.
- MBR/EBR support with visibility of partitions under Windows OS environment.
- Each partition can be operated-on independently of other partitions.
- Provides following operations on synergy platform for FAT and exFAT filesystem:
    1. Format/re-format
    2. open
    3. close
    4. write
    5. read

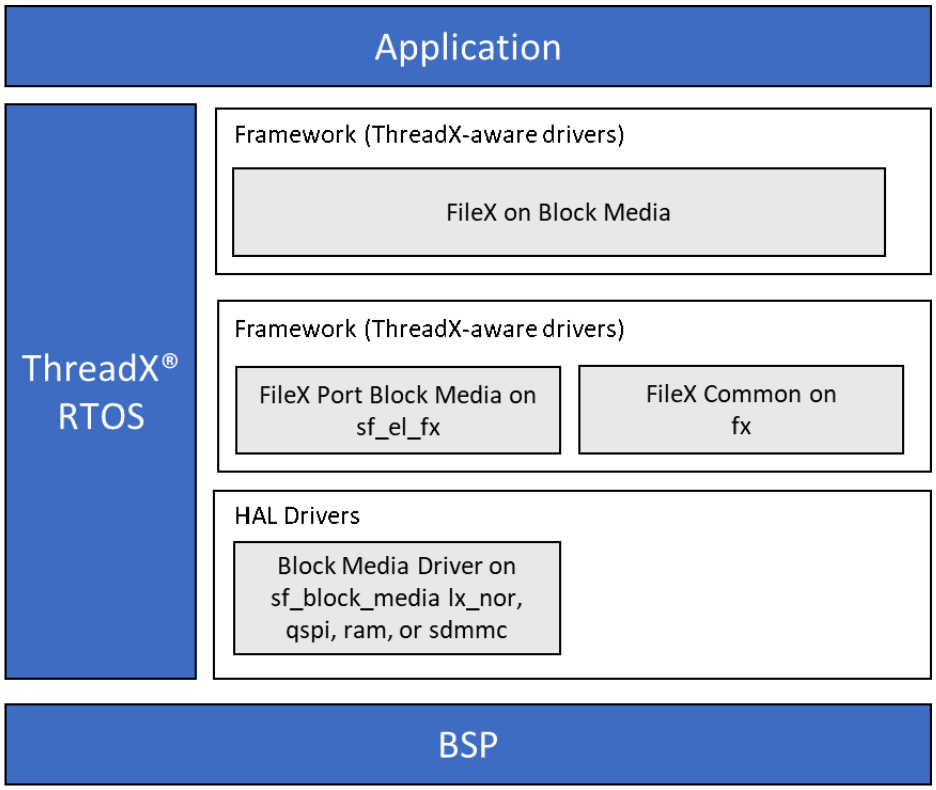


Figure 400: FileX On Block Media Framework Module Block Diagram

4.3.2.2 FileX On Block Media Framework Module APIs Overview

The FileX On Block Media Framework provides access to all the FileX APIs used to create the file system, access files and directories and manage the file system. The full set of FileX API functions, their parameters, operation and example use can be found in the FileX User's Manual. Some examples are shown below to provide some background for better understanding the rest of the descriptions of the FileX on Block media framework module.

FileX On Block Media Framework Module API Summary (Selected Examples Only)

Function Name	Example API Call and Description
fx_directory_attributes_read	fx_directory_attributes_read(&my_media, "mydir", &attributes); Reads the directory's attributes from the specified media.
fx_file_open	fx_file_open(&my_media, &my_file, "myfile.txt", FX_OPEN_FOR_READ); Open "myfile.txt" file for read.
fx_file_create	fx_file_create(&my_media, "myfile.txt"); Create file with name "myfile.txt"
fx_media_read	fx_media_read(&my_media, 22, my_buffer); Read the logic sector (in this example from sector 22) from media specified by &my_media, and places it into the buffer my_buffer.

Note

For more complete descriptions refer to the FileX User's Manual available as described in the Reference Section later in this document.

4.3.2.3 FileX On Block Media Framework Module Operational Overview

- FileX is a complete File Allocation Table (FAT) and exFAT format media and file management system for deeply embedded applications.
- FileX supports an unlimited number of media devices at the same time, like sdmmc, FLASH managers, RAM disks & Multiple other physical devices
- FileX is highly optimized for both size and performance.
- FileX provides support to following media & file operations for single and multiple partitions on the synergy platform: Format/re-format, open, close, write, read
- The user will able to view the SSP created FAT and exFAT partitions and files in windows OS environment.
- In fileX, when the exFAT support is enabled, the FAT functionalities would still be available

For single partition:

- FileX supports contiguous file allocation and FAT12, FAT16 FAT32 & exFAT formats.
- All underlying ssp block media ie sf_block_media_lx_nor, sf_block_media_sdmmc sf_block_media_qspi & sf_block_media_ram are supported

For multiple partitions:

- FileX supports the creation of any number of partition on given single physical media
- FileX supports only FAT16, FAT32 and exFAT formats.
- The following underlying block medias are supported: sf_block_media_lx_nor & sf_block_media_sdmmc.

FileX On Block Media Framework Module Important Operational Notes and Limitations

FileX On Block Media Framework Module Operational Notes

For Single Partition (Uses total available memory size):

The media must be formatted to either a FAT12, FAT16, FAT32 or exFAT file system before it can be opened, And media Format can be done prior to inserting the media (ie in a windows OS environment) or at project run time (using the "Format media" option in fileX properties in configurator).

For multi-partition:

- The following media partition format are supported: FAT16, FAT32 or exFAT file system, and the multiple partition must be created in synergy platform first before it can be used in windows environment.
- MBR sector i.e. sector 0 of the media must be erased (to erase all the existing partitions in the media), before creating new partitions in the media.
- At least one sector must be left before each EBR partition offset to accommodate the EBR sector, and at least one sector must be left before the first partition to accommodate the MBR sector.
- The auto-initialization for all the partitions must be disabled, and the partitions must be formatted and opened in the main entry function in application.

FileX On Block Media Framework Module Limitations

For Single partition:

- The exFAT file system is not supported on the sf_block_media_qspi framework (note: the user can use the sf_block_media_lx_nor instead to use the same qspi flash memory)
- For certain "total sectors" values in fileX properties, the media format in SSP application returns success, but the formatted media fails to open in the SSP application and in the windows OS environment (the user workaround for this is to increase the "Sector per cluster" value in filex properties).
- For exFAT single partition, If the block media is partially exfat formatted (ie fileX configurator option "total_sectors"<the total sectors in block media), the sd card media opens fine in the board, file operation in the board also works fine, but the media open fails in a PC windows environment (the user workaround for this is to filex format the media in the multiple of 1gb or the complete block media)

For Multiple partition:

- Partitions that are created, formatted, deleted or re-sized in any OS (for example the sd card partitions created/formatted in windows environment), when this media partitions is used with synergy platform, the SSP application will give undesired results
- Interface to below block media drivers is not supported:
    - sf_block_media_qspi (Users can alternately use sf_block_media_lx_nor)
    - sf_block_media_ram
- Auto initialization option in fileX properties is not supported, and must be disabled.
- USBX mass storage is not supported.
- The "Bytes per sector" in fileX properties is only supported for default value ie 512 bytes

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.2.4 Including the FileX On Block Media Framework Module in an Application

This section describes how to include the FileX on Block Media Framework module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

Following steps should be followed by the user for Single Partition (Used total available memory size): partitions:

To add the FileX on Block Media Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

FileX On Block Media Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_fx_media0FileXonBlock Media	Threads	**New Stack**> **X-Ware** > **FileX on Block Media**

When the FileX on Block Media Framework module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or

common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

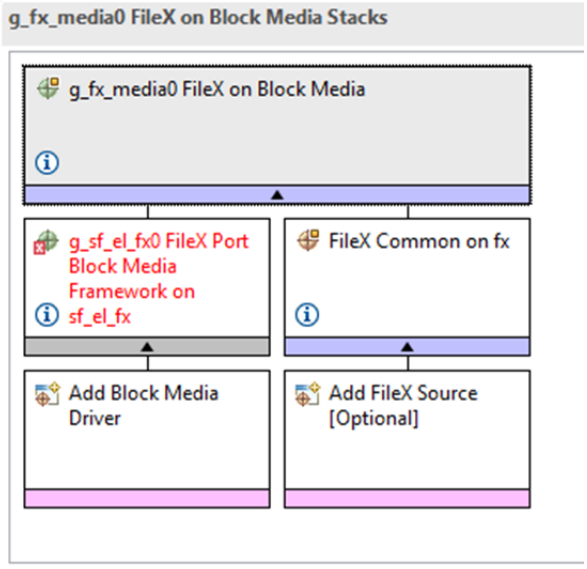


Figure 401: FileX on Block Media Stacks

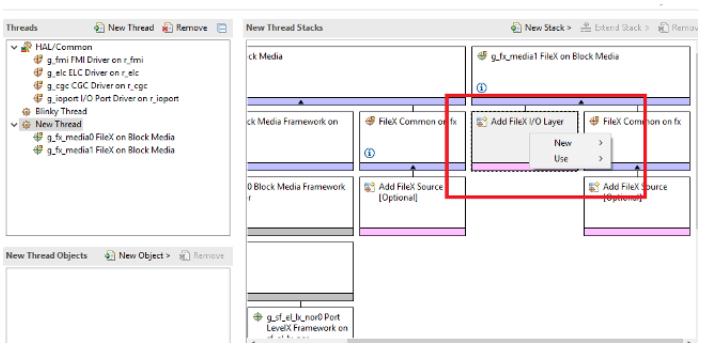
The Add Block Media Driver block must be filled in prior to generating the Synergy project code. There are multiple selections possible depending on the physical media targeted. Currently available media include RAM, LX NOR, QSPI, and SDMMC. Options may be different depending on what your target board and associated board support package (BSP) supports.

To enable the exFAT file system support in SSP, the user has to add the fileX source files to the project, and enable the “FileX Source" -> “exFAT support" option in the fileX source configurator properties, and re-build the FileX project.

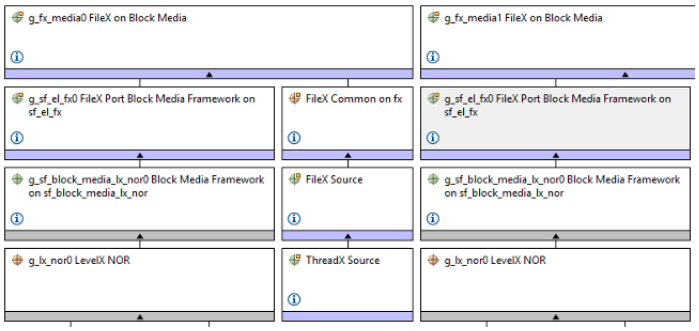
The following steps should be followed by the user for multiple partitions on the same/different media type:

1. Select FileX from X-Ware
2. Select FileX on Block Media
3. Select Block media framework - sf_block_media_lx_nor or sf_block_media_sdmmc. For example, FileX instance g_fx_media0 is created and it selects lx_nor media type.
4. Select the second Filex media instance. For example,g_fx_media1 created. For example, g_fx_media0 and g_fx_media1 represent the two partitions.

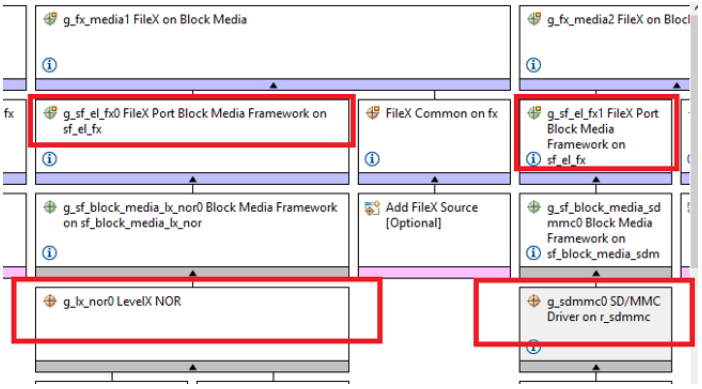
5. From the second instance and onward, selecting SF_EL_FX module instance gives two options **New** and **Use.**



6. **Use:** If the user selects **Use**, the partition will be made on the media previously selected by FileX media instance (g_sf_el_fx0 selected lx_nor). For example, g_fx_media0 and g_fx_media1 represent partitions on the same media type lx_nor.



7. **New:** If the user selects New, a new instance of SF_EL_FX is created. The user will have the choice to select the different media type. If the new instance of SF_EL_FX is chosen, a different media has to be selected for making the partition. For example: Select a third FileX media instance. Here g_fx_media2 is created. Here g_fx_media0 and g_fx_media1 have partitions on same media type with same instance of SF_EL_FX (g_sf_el_fx0). For g_fx_media2, a different instance of SF_EL_FX is selected (g_sf_el_fx1). g_fx_media2 selects a new instance of SF_EL_FX module (g_sf_el_fx1) which selects SDMMC block media framework. Here, g_fx_media0 and g_fx_media1 have partitions on media type qspi memory. g_fx_media2represents the partition on SDMMC framework.)



8. For g_fx_media3, the user has the choice to select **Use** (for SF_EL_FX module) or **New** for a different media type.

4.3.2.5 Configuring the FileX On Block Media Framework Module

The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the FileX On Block Media Framework Module

ISDE Property	Value	Description
Name	g_fx_media0	Module name.
Format media during initialization	Enable, Disable Default: Disabled	Select to automatically format the media during initialization.
File System is on block media	True, False Default: True	This setting will use code that reads sector size and count from the media device before formatting, if true. If set to false, the format function will use the sector size and count from configuration below.
Formatting Options		

Volume Name	Volume 1	This is the volume name to use when formatting the media device. It should be no longer than 11 characters.
Number of FATs	1	Number of File Allocation Tables.
Directory Entries	Default: 256	Number of directory entries in the root directory.
Hidden Sectors	0	Number of hidden sectors.
Total Sectors	3751936	Total number of sectors in the media. *Note* *When "File System is on Block media" property is set to false, enter appropriate total sector.*
Bytes per Sector	512	Sector size to format, when using the format function. "File System is on SDMMC" must be disabled for this value to be used. *Note* *When selected QSPI as media, Bytes per sector must be at least 4096.*
Sectors per Cluster	1	Sectors per cluster. This value is not used if "File System is on SDMMC" disabled.
Volume Serial Number	12345	Serial number to be used for this volume. Only used if Format media during initialization is enabled. Used in exFAT media format only.
Boundary unit	128	Physical data area alignment size, in number of sectors. Only used if Format media during initialization is enabled. Used in exFAT media format only.

Working media memory size	512	This is the number of bytes FileX uses to read and write to the media. It should be at least the size of one sector. *Note* *'Working media memory size' must be equal to or greater than 'Block size of media in bytes'. When selected QSPI as media, then working media memory size must be at least 4096 or greater.*
Name of generated initialization function	fx_media_init0	Name of the initialization function that is generated by SSP.
Auto Initialization	Enable, Disable Default: Enable	Select to enable or disable auto initialization.

Note

For multi partition Auto initialization and media format during initialization should be disabled.
For multi partition, "Bytes per sector" in filex properties is only supported for default value ie 512 bytes
The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

For Multi partitions:

sf_el_fx user configuration settings are require for multi-partition operation.

ISDE Property	Value	Description
Name	g_sf_el_fx0	Module name.
Partition info callback	A callback can be NULL or the name of the user-defined function.	If partitions exist and the system got reset, user callback specifies the partition offset/base address. This callback is called while opening the partition.
Total partitions	>1	The number of partitions users wanted to make on the media.

1. **Partition info callback:** Partition will be open as per user requirement by setting partition info callback to user define function or by setting it to NULL value it will open partitions in linear manner.
2. **Total partitions:** Total partition specifies the desired number of partitions. Values 0 and 1 represent a single partition. Any value more than 1 indicates the required partition.

Configuration Settings for the FileX On Block Media Framework Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Block Media Framework on sf_block_media_sdmmc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_block_media_sdmmc0	Module name.
Block size of media in bytes	512	Specify the size of a block in bytes.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SDMMC HAL Module on r_sdmmc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_sdmmc0	The name to be used for SDMMC module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	Select the channel.
Media Type	Embedded, Card Default: Embedded	Media is a card or an embedded device. This allows to firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 Bit, 4 Bits, 8 Bits Default: 4 Bits	Data bus with as defined by hardware interface. (8 Bits for eMMC only)
Block Size	512	Block size selection.
Card Detection	Not Used, CD Pin Default: CD Pin	Card detection selection.

Callback	NULL	(Not required if using Filex) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR
Access Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Access interrupt priority selection.
Card Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Card interrupt priority selection.
DTC Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	DTC interrupt priority selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

FileX Port Block Media Framework Module Clock Configuration

The SDHI uses the PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO or eMMC device obtained at media device initialization.

FileX Port Block Media Framework Module Pin Configuration

Use the e^2 studio pin configurator to configure the I/O pins for the target media peripheral. The following description applies to the SDHI peripheral used for SDMMC media. Other media will require similar pin settings, and these are easily determined using their associated pin selection sequence in the Synergy Configurator. Consult the appropriate HAL driver section in the SSP User's Manual for additional details.

For SDMMC media, the drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table provides an example selection for the module pins.

Note

The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for the SDHI Peripheral

Resource	ISDE Tab	Pin selection Sequence
SDHI	Pins	Select **Peripherals**> **Storage:SHDI**> **SDHI0**

Note

The selection sequence assumes that the SDHI0 is the desired hardware target for the driver.

Pin Configuration Settings for the SDHI Peripheral

Pin Configuration Property	Settings	Description
Operation Mode	Disabled, Custom, SD_MMC 1 bit SD_MMC 4 bit MMC 8 bit Default: Custom	Select mode as per application.
CLK	None, P413 Default: P413	Clock pin.
CMD	None, P412 Default: P412	Command pin.
DAT0-7	None, PXXX Default: PXXX	Data pin.
CD	None, P903 Default: P903	Card detection pin.
WP	None, P414 Default: P414	Card write protection pin.

Note

The example values are for a project using the Synergy S7G2 MCU Group and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.3.2.6 Using the FileX on Block Media Framework Module in an Application

The steps in using the FileX on Block Media Framework module in a typical application are:

1. Initialize the media using the FileX API fx_system_initialize (FileX on Block Media calls it automatically if **Auto Initialization** property is set to Enabled in **FileX Common on fx**)
2. Optionally, format/create a FAT media using the FileX API "fx_media_format", (or)

format/create a exFAT media using the FileX API "fx_media_exFAT_format" (FileX on Block Media automatically formats the media using the auto-generated initialization function if its **Format media during initialization** property is set to Enabled).
3. Open the media using the FileX API fx_media_open (FileX on Block Media opens the media automatically if its **Auto Initialization** property is set to Enabled)
4. Create and delete files and directories as required using one of the FileX APIs (for example, fx_file_create, fx_file_delete, fx_directory_create, fx_directory_delete)
5. Read from and write to files on the media as required using one of the FileX API (for example, fx_file_read or fx_file_write)
6. Read from and write to the media directly as required using one of the FileX API (for example, fx_media_read or fx_media_write)
7. Close the physical media using the FileX API fx_media_close

Note

After a successful fx_media_open call, all FileX file and directory related APIs can be used. Refer to the FileX User Guide for documentation on all available functions.

These common steps are illustrated in a typical operational flow in the following figure:

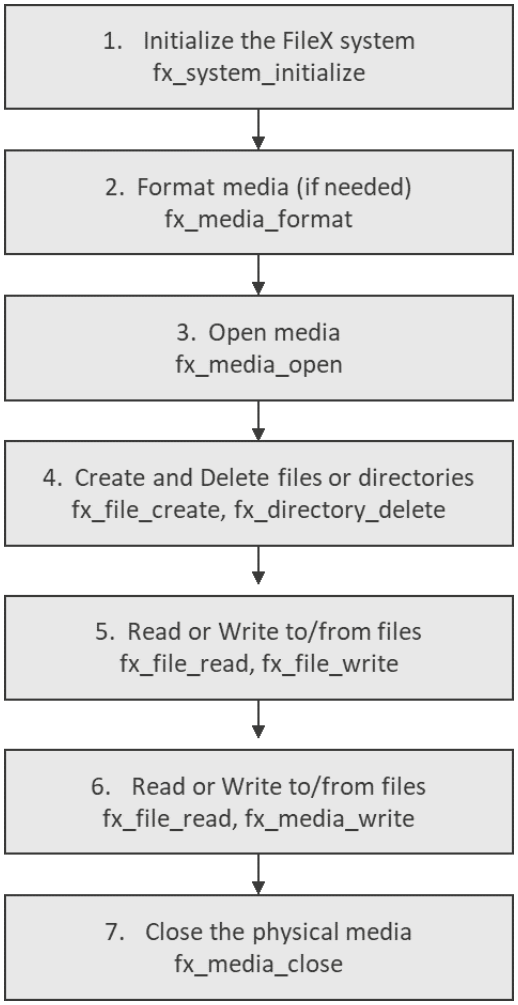


Figure 402: Flow Diagram of a Typical FileX On Block Media Framework Module Application

4.3.3 FileX Source

4.3.3.1 FileX Source Component Module Introduction

The purpose of this document is to provide an easy reference for the FileX source component in e^2 studio. The properties are explained in greater detail than the footer comment supplied with each property. Context specific usage is included to help understand when to change the default values. This document should make it easier to use the FileX source component without having to cross reference with the Azure RTOS FileX User Guide, and help the developer become familiarized, more quickly, with FileX features.

4.3.3.2 When to Include the FileX Source Component

Adding the FileX source component enables the developer in the Synergy configurator environment to customize the FileX library, change values from default settings, and enable or disable certain features. Otherwise, they must use the prebuilt FileX library. In most projects beyond the simplest, the developer will typically want to customize their FileX environment. Note that the ThreadX source component is automatically added whenever FileX is added as a higher-level source component.

Without adding the FileX source component, the e^2 studio configurator will use a prebuilt library with the FileX default settings.

4.3.3.3 Adding the FileX Source Component

In the e^2 studio configurator, add the ThreadX source component by selecting any thread from the Threads list and pressing the **New Stack** button and navigating the menu to **X-Ware** > **FileX**> **FileX Source**. Often the FileX source is available as an option when a high-level framework is created. For example, the FileX source module is available as an option for the FileX on USB Mass Storage framework module as seen in the below thread stack diagram.

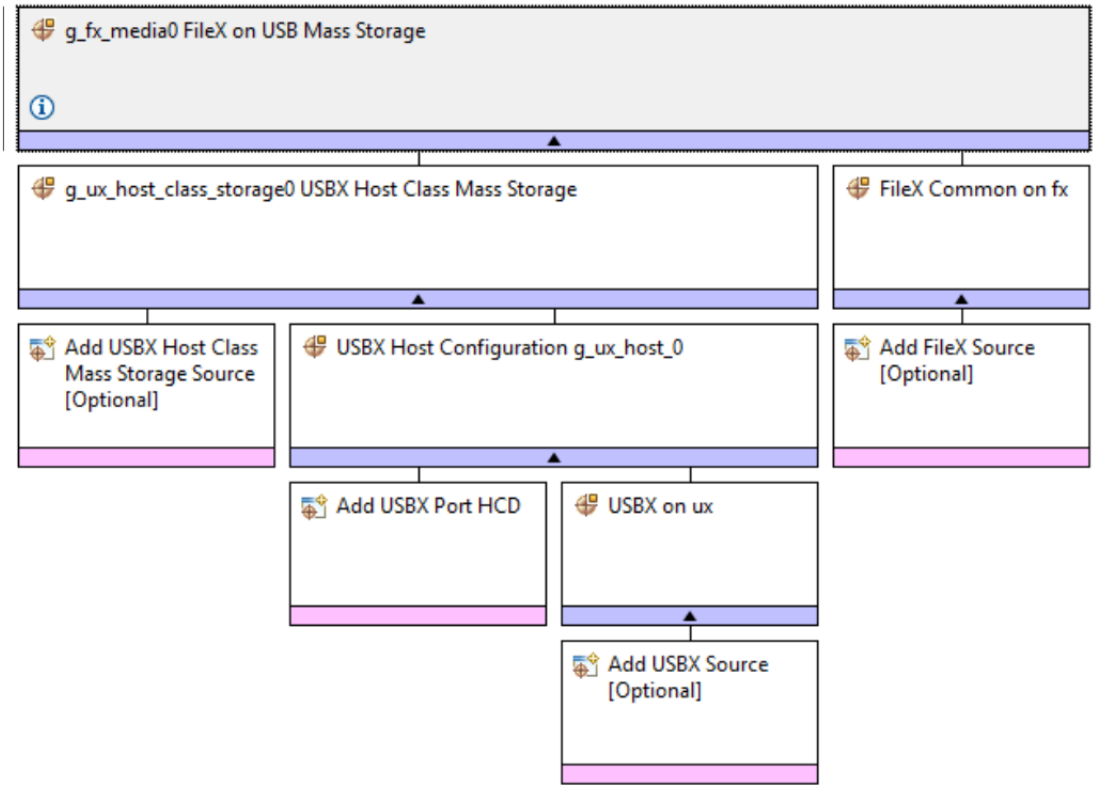


Figure 403: FileX Source Module Stack

4.3.3.4 Changing the FileX Source Component Properties

After changing FileX property settings, the developer must click on the Generate Project Content button to update the project configurator in e^2 studio and then the FileX library **must** be rebuilt (for example, rebuild the project). Simply changing a property (or applying a #define in the preprocessor list) without rebuilding the project will not affect any change since e^2 studio will use the previously built library.

The default settings are often the choice needed for the most common use cases.

4.3.3.5 FileX Source

The properties of the FileX Source component are given in the order they appear in the properties window of the Synergy configurator.

Error Checking – default value enabled – Generally enabled during development and debugging phase, and disabled when a release version is being built. When enabled, FileX include error checking services that will check input and other parameters before calling the actual API. Some of the things it checks for are:

- NULL pointer input
- Invalid non-pointer parameters, such as an invalid file or directory names.
- Required configurable option must be enabled. For example, performance information must be enabled to call the get services.
- The data structure IDs must match what is expected. For example,
    - file_ptr -> fx_file_id != FX_FILE_ID // check the file instance structure
- Size of the data structure, the FileX file for example, matches the size of the data structure

in the FileX library.

These last two checks guard against an application using a different version of the FileX library than the application is using.

Disabling the FileX error checking API results in improved performance (as much as 30%) and smaller code size.

Max Long Name Len – Maximum Long Name Length – default value not displayed, 33 characters is used – Defines the maximum size of long file names supported by FileX. The default value is 33. The minimum value is 13 and the maximum value is 256.

Max Last Name Len – Maximum Last Name Length – default value not displayed, 256 characters is used – Defines the maximum size of last opened file names supported by FileX. The default value is 256. The minimum value is 13 and the maximum value is 256. Must be as large as or larger than Max Long Name Len.

Max Sector Cache – Maximum Sector Cache Size - default value not displayed, 256 is used – Defines the maximum number of logical sectors that can be cached by FileX. The cache memory supplied to FileX at fx_media_open determines how many sectors can actually be cached. Minimum value is 2, all other values must be a power of 2.

Fat Map Size – FAT Map Size - default value not displayed, 128 is used – Defines the size in bytes of the bit map used to update the secondary FAT sectors. Larger the value result in fewer unnecessary secondary FAT sector writes. Minimum value is 1, no maximum value.

Max Fat Cache – Maximum FAT Cache - default value not displayed, 16 is used – Defines the number of entries in the internal FAT cache. The minimum value is 8, all values must be a power of 2.

Update Rate (seconds) - **default value not displayed, 10 seconds used** - Specifies rate at which the system time in FileX is adjusted. The default value 10 means that the FileX system time is updated every 10 seconds.

__Max exFAT Cache Size - default value not displayed, 512 is used. -__ Defines bitmat cache size for exFAT. Size should be minimum one sector size and maximum 4096. For applications using multiple media devices with varying sector size, the value should be set to the size of largest sector size.

No Timer – default value disabled – When enabled, FileX is built without update to the time parameters. Eliminates the ThreadX timer setup to update the FileX system time and date. Doing so causes default time and date to be placed on all file operations.

Single Thread – default value disabled – When enabled, FileX is running in a single-threaded environment and does not need thread protection. Eliminates ThreadX protection logic from the FileX source. It should be used if FileX is being used only from one thread or if FileX is being used without ThreadX.

Don't Update Open Files – default value disabled – When enabled, FileX does not update already opened files.

Media Search Cache – default value enabled – When enabled, a cache is used for optimization when searching for open media. Disabling this option will remove this optimization, reducing code size and memory footprint at the expense of performance.

Direct Data Read Cache Fill – default value enabled – When enabled, data sector reads are cached for faster access. Disabling this feature will reduce code size and memory footprint at the expense of performance.

Media Statistics – default value enabled – Determine if media statistics are kept and gathered. When disabled, no media statistics are available. This improves performance slightly and reduces code size; each instance of the FX_MEDIA_STRUCT structure is considerably smaller.

Single Open Legacy – default value disabled – When enabled, legacy single open logic for the same file is used. This may be necessary to make the fx_file_open behave in the same way as older versions of FileX, when migrating old FileX application code.

Rename Path Inherit – default value disabled – When enabled, renaming inherits path information. In other words, prepend the path in the new file name, then override the old file name with the new one on the renamed file.

No Local Path – default value disabled – When enabled, the local path logic is disabled. When disabled, a local path is kept for each thread; all operations performed with a relative path, will be relative to this local path.

Fault Tolerant Data – default value disabled – When enabled, data sector write requests are flushed immediately to the driver. This will increase the likelihood that data is not lost in case of power loss or a reset, but it will impact performance.

Fault Tolerant – default value disabled – When enabled, system sector write requests (including FAT and directory entry requests) are flushed immediately to the driver. This will increase the likelihood that data is not lost in case of power loss or a reset, but it will impact performance.

64-bit LBA – default value enabled – When enabled, 64-bits sector addresses are used in the I/O driver. This allows bigger media and bigger files.

Fault Tolerant Service – default value disabled – Enables or disables the FileX fault tolerant service. The FileX Fault Tolerant Module is designed to prevent file system corruption caused by interruptions during the file or directory update. For example, when appending data to a file, FileX needs to update the content of the file, the directory entry, and possibly the FAT entries. If this sequence of update is interrupted (such as by a power glitch, or if the media is ejected in the middle of the update), the file system is in an inconsistent state, which may affect the integrity of the entire file system, leading towards corruption of other files.

Fault Tolerant Boot Index – default is 16 – Defines byte offset in the boot sector where the cluster for the fault tolerant log is. By default, this value is 116. This field takes 4 bytes. Bytes 116 through 119 are chosen because they are marked as reserved by FAT 12/16/32/exFAT specification.

Fault Tolerant Minimal Cluster Size – default is 3072 – Defines the requirement for minimal size of a cluster in bytes. It must be a multiple of the sector size. The default value is 3072, which works with the worst case for long file renaming.

exFAT Support – default value disabled – When enabled, the logic for handling the exFAT file system is enabled in FileX source code

Standalone enable – default value is disabled – When enabled, Filex will be used in standalone mode (without ThreadX).

Disable cache – default value is disabled – When enabled, cache is disabled.

Disable file close – default value is disabled – When enabled, file close is disabled.

Disable fast open – default value is disabled – When enabled, fast open is disabled.

Disable force memory operation – default value is disabled – When enabled, force memory operations are disabled.

Disable build option – default value is disabled – When enabled, build option is disabled.

Disable one line function – default value is disabled – When enabled, one line function is disabled.

Disable fat entry refresh – default value is disabled – When enabled, FAT entry refresh is disabled.

Disable consecutive detect – default value is disabled – When enabled, consecutive detect is disabled.

4.3.3.6 FileX Fault Tolerant Module

When an application writes data into a file, FileX updates both data clusters and system information. These updates must be completed as an atomic operation to keep information in the file system coherent. For example, when appending data to a file, FileX needs to find an available cluster in the media, update the FAT chain, update the length filed in the directory entry, and possibly update the starting cluster number in the directory entry. Either a power failure or media ejection can interrupt the sequence of updates, which will leave the file system in an inconsistent state. If the inconsistent state is not corrected, the data being updated can be lost, and because of damage to the system information, subsequent file system operation may damage other files or directories on the media.

The FileX Fault Tolerant Module works by journaling steps required to update a file before these steps are applied to the file system. If the file update is successful, these log entries are removed. However, if the file update is interrupted, the log entries are stored on the media. Next time the media is mounted, FileX will detect these log entries from the previous (unfinished) write operation. In such cases, FileX can recover from a failure by either rolling back the changes already made to the file system, or by reapplying the required changes to complete the previous operation. In this way, the FileX Fault Tolerant Module maintains file system integrity if the media loses power during an update operation.

Note

The FileX Fault Tolerant Module is not designed to prevent file system corruption caused by physical media corruption with valid data in it.

After the FileX Fault Tolerant module protects a media, the media must not be mounted by anything other than FileX with Fault Tolerant enabled. Doing so can cause the log entries in the file system to be inconsistent with system information on the media. If the FileX Fault Tolerant module attempts to process log entries after the media is updated by another file system, the recovery procedure may fail, leaving the entire file system in an unpredictable state.

The FileX Fault Tolerant feature is available to all FAT file systems supported by FileX, including FAT12, FAT16, FAT32, and exFAT. To enable the fault tolerant feature, FileX must be built with the option "Fault tolerant service" enabled. At run time, the application starts the fault tolerant service by calling fx_fault_tolerant_enable() immediately after the call to fx_media_open. After fault tolerant is enabled, all file write operations to the designated media are protected. By default, the fault tolerant module is not enabled.

The application needs to make sure the file system is not being accessed prior to fx_fault_tolerant_enable() being called. If application writes data to the file system prior to fault tolerant enable, the write operation could corrupt the media if prior write operations were not completed, and the file system.

The FileX fault tolerant log takes up one logical cluster in flash. The index to the starting cluster number of that cluster is recorded in the boot sector was not restored using fault tolerant log entries. For further details on the log format refer to the FileX user guide.

4.3.3.7 About exFAT Support

To enable the exFAT file system support in SSP, the user has to add the fileX source files to the project, and enable the “FileX Source" -> “exFAT support" option in the fileX source configurator properties, and re-build the FileX project.

4.3.4 GUIX Port

4.3.4.1 GUIX Synergy Port Framework Introduction

The SF_EL_GX (GUIX Port) module is the Azure RTOS GUIX adaptation layer for Synergy MCU groups, which have graphics engines GLCDC, DRW (2DG engine) or a JPEG decode engine. The API supports graphics hardware engine setup for GUIX and supports graphics rendering and displaying accelerated by hardware engines. The module defines full-set of GUIX low-level display driver functions which draw graphics accelerated by the DRW (2DG engine) or the JPEG or displays graphics with the GLCDC (See the GUIX User Guide Chapter 5: GUIX Display Drivers). The module encourages the hardware acceleration for graphics rendering, but also allows software processing without hardware support.

Supported and Unsupported Features

The following GUIX features are supported in SSP:

- RGB565 Pixel Format
- CLUT8 Pixel Format
- ARGB888 Pixel Format
- ARGB8888 Pixel Format
- Rotation Of Screen: Supported in SSP v2.0.0
- Dynamic Loading of Binary Resources
- Hardware and Software JPEG Support

The following features are not supported:

- ARGB4444 Format
- Multiple Canvases: GUIX allows use of multiple canvases but SSP supports only a single canvas

GUIX Synergy Port Framework Module Features

- Adapts GUIX on top of the SSP

- Attaches the SSP Display Interface driver to GUIX Display Driver Interface
- Allows GUIX to draw widgets accelerated by the Synergy D2W (2DG) engine
- Allows GUIX to draw widgets accelerated by the Synergy JPEG engine
- Supports double-buffer toggling control for screen transitions without tearing
- Supports screen rotation (90/180/270 degree)
- Supports various output color formats
    - 32bpp (ARGB8888, RGB-888)
    - 16bpp (RGB565)
    - 8bpp (8bit Palette (CLUT))
- Supports user callback functions

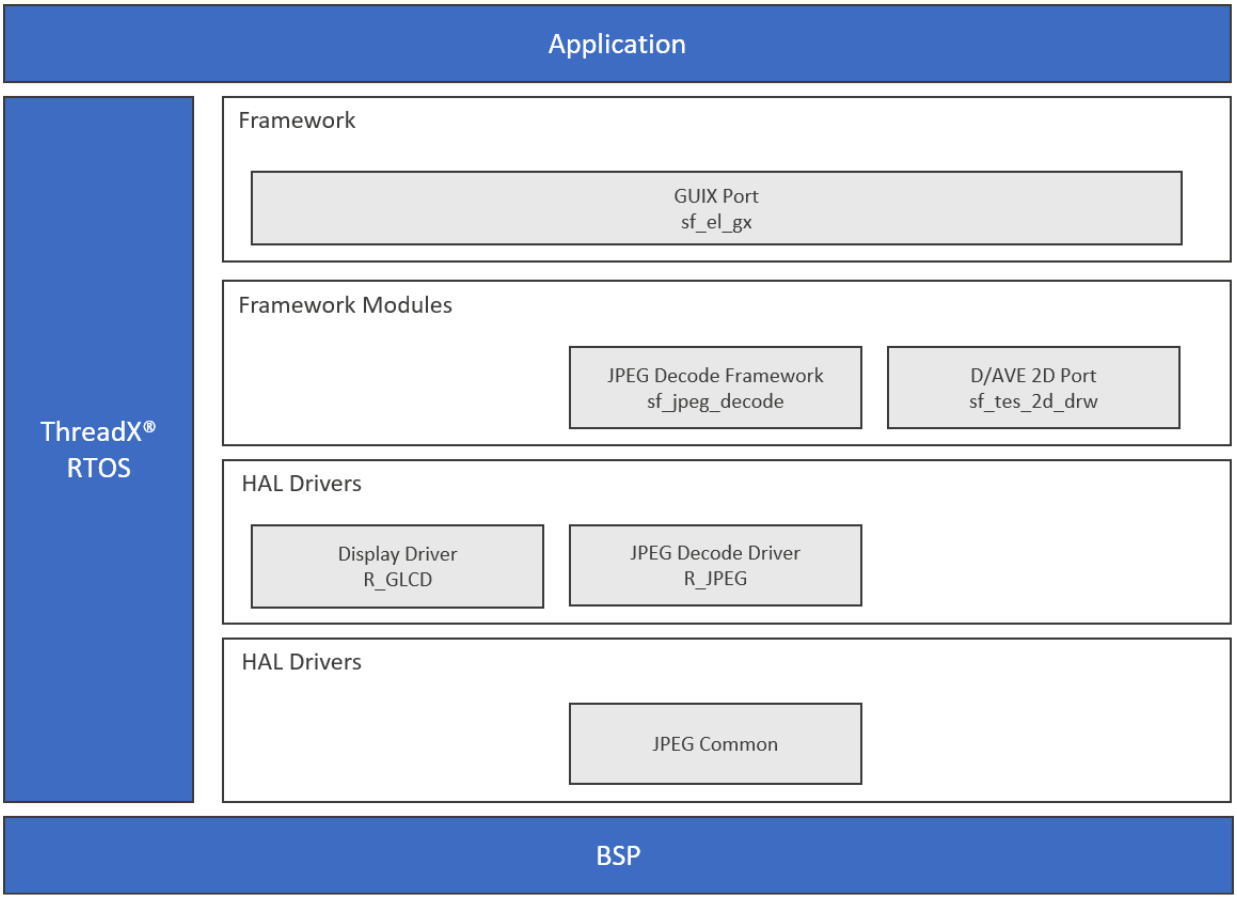


Figure 404: GUIX Synergy Port Framework Module Block Diagram

4.3.4.2 GUIX Synergy Port Framework Module APIs Overview

The GUIX Synergy Port Framework defines APIs for opening, closing, setup and initialization. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

GUIX Synergy Port Framework Module API Summary

Function Name	Example API Call and Description

open	g_sf_el_gx.p_api->open(g_sf_el_gx.p_ctrl, g_sf_el_gx.p_cfg); Opens the SF_EL_GX Module. The API can only be called from a thread. The API passes the configuration pointer to define low-level graphics device drivers and frame buffers and register the user callback function. This function does not actually initialize low-level drivers. Instead, the API setup initializes the low-level drivers.
close	g_sf_el_gx.p_api->close(g_sf_el_gx.p_ctrl); Closes the SF_EL_GX Module. This API closes the low-level drivers. Normally, the API is not called since GUIX will not be closed once initialized.
versionGet	g_sf_el_gx.p_api->versionGet(&version); Returns the version of the Module in the version pointer.
setup	gx_studio_display_configure (MAIN_DISPLAY, g_sf_el_gx.p_api->setup, LANGUAGE_ENGLISH, MAIN_DISPLAY_THEME_1, &p_window_root); This is the interface to initialize low-level graphics device drivers and must be passed to GUIX through GUIX (Studio) service call gx_studio_display_configure() as the function pointer. GUIX then calls the API back and, at that moment, the API configures the SSP device drivers based on the configuration passed by open. The reason for this procedure to initialize low-level drivers is that the API has the GUIX-compliant argument (GX_DISPLAY *) type and does not allow applying the detailed configuration of the SSP graphics device drivers generated from e^2 studio.
canvasInit	g_sf_el_gx.p_api->canvasInit(g_sf_el_gx.p_ctrl, p_window_root); This is the GUIX helper API to determine the memory address of GUIX canvas. The API has an argument with (GX_WINDOW_ROOT *) type and the API provides GUIX the start address of canvas memory, which is needed for the low-level graphics device drivers to draw/display images.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.

SSP_ERR_ASSERTION	NULL pointer error happens.
SSP_ERR_IN_USE	SF_EL_GX is in-use.
SSP_ERR_INTERNAL	Error happen in Kernel service calls.
SSP_ERR_NOT_OPEN	SF_EL_GX is not opened.
SSP_ERR_TIMEOUT	A task times out (or exceeds retry limit) before completion in display driver.
SSP_ERR_D2D_ERROR_DEINT	Error occurred in D/AVE 2D driver.
GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.
SSP_ERR_INVALID_CALL	Function call was made when the driver is not in SF_EL_GX_CONFIGURED state.
SSP_ERR_D2D_RENDERING	The D/AVE 2D returns error at opening a display list buffer.
SSP_ERR_INVALID_ARGUEMENT	Invalid non-pointer (for example, parameter) input.
SSP_ERR_UNSUPPORTED	Specified color format is not supported.
SSP_ERR_D2D_ERROR_INIT	The D/AVE 2D returns error at the initialization.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.3.4.3 GUIX Synergy Port Framework Module Operational Overview

The following block diagram shows how the sf_el_gx module interfaces with the other Synergy components at the operational level. It also shows the layers at which the callback and low-level modules communicate with the rest of the system. Refer to this diagram while reading the operational overview and operational notes descriptions provided below.

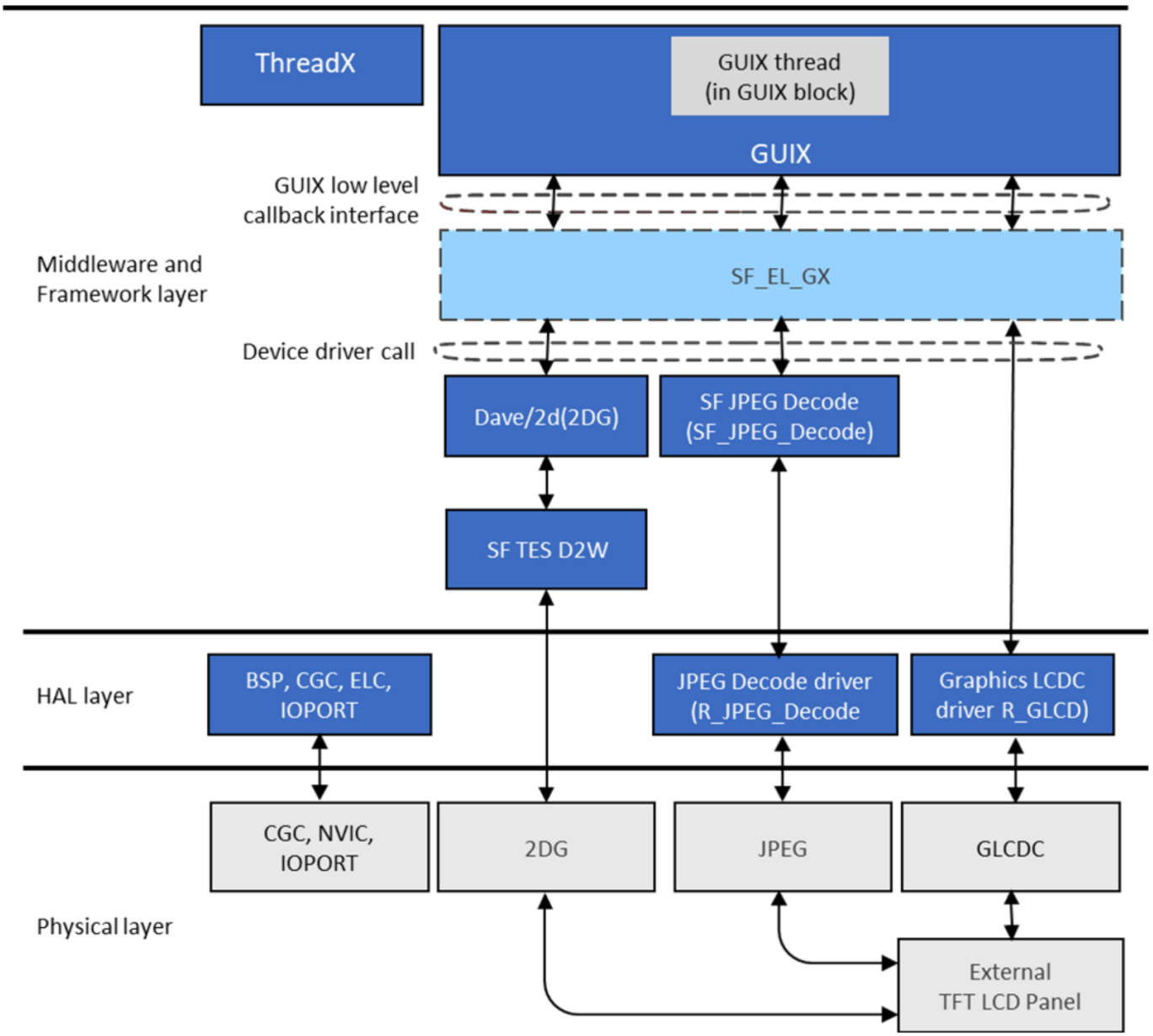


Figure 405: GUIX Synergy Port Framework Module Flow Chart

Module Initialization

The SF_EL_GX supports the Synergy graphics hardware setup, which is required to run the GUIX system. The module has a dependency with Azure RTOS GUIX™ and GUIX Studio™ generated code. The GUIX system initialization needs to follow the sequence below as a general guidance.

1. open SF_EL_GX module to initialize SF_EL_UX control block and pass module configurations.
2. Initialize GUIX Display object by GUIX Studio generated API gx_studio_display_configure. Through this API, SF_EL_GX setup API is input to GUIX and Synergy graphics hardware setup will complete. Also, the root window initialized by GUIX is output to a user application.
3. Initialize the primary memory address of a GUIX Canvas Buffer by canvasInit API.
4. Create the root window by GUIX Studio generated the gx_studio_named_widget_create API.
5. Show the root window with the GUIX gx_window_show API.
6. Start the GUIX system withe GUIX gx_system_start API.

Ping-Pong Frame Buffer Management

The SF_EL_GX module manages the buffer toggling operation in the graphics system with a ping-pong frame buffer. The figure below shows a ping-pong buffer graphics system managed by the SF_EL_GX module. The module uses GUIX and the low-level display driver functions to draw an image (2D Drawing engine(DRW) or JPEG) and display the image (DISPLAY module, for example: GLCDC). A design with a single frame buffer is also possible in the SF_EL_GX configuration. However, it is encouraged to use two frame buffers to avoid the tearing issue that could occur in a single frame buffer system.

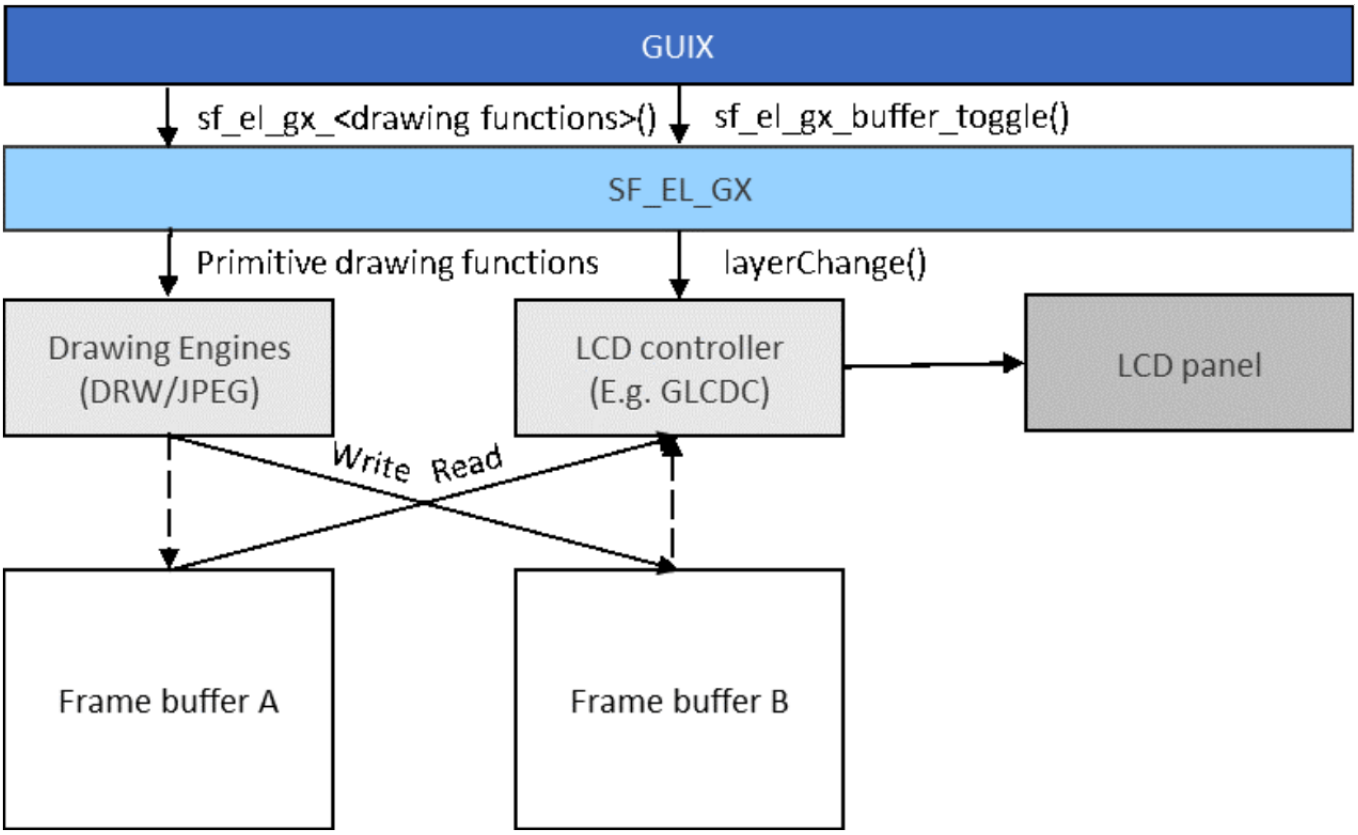


Figure 406: GUIX Synergy Port Framework Ping Pong Frame Buffer System

GUIX Synergy Port Framework Module Important Operational Notes and Limitations

GUIX Synergy Port Framework Module Operational Notes

Synergy 2D Drawing Engine Support

The module can draw graphics image accelerated by 2D Drawing engine (DRW) to get better graphics performance. Users can enable 2D Drawing engine with following configurations. The configuration is available through Synergy Configurator.

- Define GX_USE_SYNEGY_DRW (1) in gx_user.h
- Set DRW (SF_TES_2D_DRW) interrupt priority

Make sure to select **Renesas Synergy** in **Target CPU** setting in the **Configure Project** window and check **Enable Graphics Accelerator** in the **Synergy Advanced Settings** window on the Azure RTOS GUIX Studio (v5.2.8 or later). For the Pixelmap Output Format in the **Edit Pixel map** window (Right-click on **pixelmap**->**edit settings** to open the window), select **Compress Output**.

Do not select **Raw Format**. This configuration allows the GUIX Studio to generate Targa RLE formatted encoded image resource data. The 2D Drawing engine hardware can read this format and decode and draw the image on the frame buffer.

Synergy JPEG Support

The module can draw graphics image accelerated by JPEG engine to get better graphics performance if a JPEG encoded image is used as a GUIX image resource. Users can enable JPEG engine with following configurations. The configuration is available through Synergy Configurator.

- Define GX_USE_SYNEGY_JPEG (1) in gx_user.h
- Set JPEG (R_JPEG_DECODE) interrupt priority

Make sure to select **Renesas Synergy** in **Target CPU** setting in the **Configure Project** window and select **Hardware JPEG Decoder** in the **Decoder Types JPEG:** drop-down menu on the Azure RTOS GUIX Studio (v5.2.8 or later). For the Pixelmap Output Format in the **Exit Pixel map** window (Right-click on **pixelmap**->**edit settings** to open the window), select "Raw Format". This configuration allows the GUIX Studio to generate raw JPEG encoded image resource data. JPEG hardware can read this format and decode and draw the image on the frame buffer.

GUIX 5.4.0 API Compatibility Support

The Certain GUI APIs were modified post GUIX 5.4.0 to add support for disabled test colors and to improve the accuracy of certain math functions by using fixed point match parameters. To support existing applications which use the legacy APIs, configure following property in Synergy Configurator. The Legacy APIs can be used by enabling the property.

- Enable GUI Legacy API Support

The new version of these APIs should be preferred while creating a new application. In order to use new version of these APIs, the GUIX Source should be included in the project and the above property should be disabled.

GUIX Canvas Buffer

With SSP v2.0.0 or later,, the canvas buffer is required only for 180 (or Flip) degree screen rotation. For 180 screen rotation operation, the GUIX first draws the image on a Canvas buffer and then image is flipped and stored in the frame buffer. The size of GUIX Canvas Buffer must be exactly same as a frame buffer for the DISPLAY module. Note that, the use of a GUIX Canvas impacts to the graphics performance because of additional graphics image processing being required. Therefore, GUIX Canvas buffer should be only used if the 180 degree screen rotation is required. Otherwise set NULL to sf_el_gx_cfg_t::p_canvas to let GUIX draw an image directly to frame buffers.

Screen Rotation

The module supports the screen rotation. Supported rotation angle is either of 90, 180, 270 degree in clockwise way and must be set in the GUIX Studio. With SSP v2.0.0 or later, the canvas buffer is not required for the 90 and 270 degree screen rotation and must not be set. However, to enable the 180 degree screen rotation feature, a GUIX Canvas Buffer is still required and should be set in sf_el_gx configuration property. In this case, GUIX draws screen update on a canvas first and then GUIX Port processes the screen copy to a frame buffer with rotating the image by 180 degree. Dynamic screen rotation is not supported. If *Synergy 2D Drawing Engine Support* is enabled (GX_USE_SYNEGY_DRW = 1), the 180 screen rotation is processed by 2D Drawing engine with texture mapping. If not enabled (GX_USE_SYNEGY_DRW = 0), it is processed by software copy.

Note that starting with the SSP 2.0.0 the configuration option sf_el_gx_cfg_t::rotation_angle is not available. The rotation angle must be set in GUIX Studio project. Make sure to select rotation angle**(**either **None, CW, CCW or FLIP**) in **Rotation** setting in the **Configure Project** window on the Azure RTOS GUIX Studio (v6.1.6.2 or later). The earlier 90 degree Counter Clock wise rotation angle option in sf_el_gx_cfg_t::rotation_angle is now replaced with CCW option in GUIX Studio. Similarly, earlier 180 and 270 degree rotation options are replaced with FLIP and CW options in GUIX Studio. The (sf_el_gx_cfg_t::p_canvas = non-NULL value) is allowed for rotation angle other than 180 degree, but should not be done. This configuration just consumes extra bus bandwidth for screen image copy from a GUIX Canvas buffer to frame buffers. Therefore, set NULL to sf_el_gx_cfg_t::p_canvas to not use a GUIX Canvas buffer.

Size of JPEG Work Buffer

The JPEG work buffer trades off the JPEG decode speed against the buffer size. When a widget on the screen is formatted in JPEG, the JPEG work buffer is used as a temporary storage memory to create the decoded image. If the buffer size is not large enough for decoding an entire image, JPEG decoding is performed in the output buffer streaming mode. BitBLT operation by 2D Drawing engine decodes a piece of JPEG raster image in the buffer, then transfers it to the frame buffer. The minimum size of JPEG work buffer is {(The number of pixels in the horizontal line) x (bpp (bytes per pixel) of the display format) x 8 (lines)}. For instance, if the decoded image is 800 pixels in a horizontal line and RGB565 format, the number is 800 x 2 x 8= 12 800 (byte). If the buffer size was smaller than this number, JPEG decoding will not be processed. To get better throughput, parameter "Size of the JPEG Work Buffer" should be set as much as larger because it improves the JPEG decode throughput. The JPEG output buffer streaming mode repeats partial JPEG decode operations and the repletion comes to be overhead.

D/AVE 2D Buffer Cache

The D/AVE 2D buffer cache can be enabled or disabled through following configuration in Synergy Configurator. Disable it when images with high resolution and 32 bit ARGB8888 color format are used.

- D/AVE 2D Frame Buffer Cache (Valid if D/AVE 2D Drawing Engine is enabled)

Screen Tearing in Single Buffer Designs

Screen tearing is a visual artifact in video display where a display device shows information from multiple frames in a single screen draw. In general, a system with a single frame buffer can cause the screen tearing issue on a LCD panel. The module allows users to have single frame buffer (set NULL to sf_el_gx_cfg_t::p_framebuffer_b) but does not care for the screen tearing. It is recommended to have a ping-pong frame buffer system to consist of two frame buffers.

GUIX Synergy Port Framework Module Limitations

SF_EL_GX is only applicable for the Synergy MCU with GLCDC (mandatory), 2D Drawing engine or JPEG engine (optional)

Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.4.4 Including the GUIX Synergy Port Framework Module in an Application

This section describes how to include the GUIX Synergy Port Framework module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the GUIX Synergy Port Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

GUIX Synergy Port Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_gx0 GUIX Port on sf_el_gx	Threads	**New Stack> Framework> Graphics> GUIX Port on sf_el_gx**

When the GUIX Synergy Port Framework module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

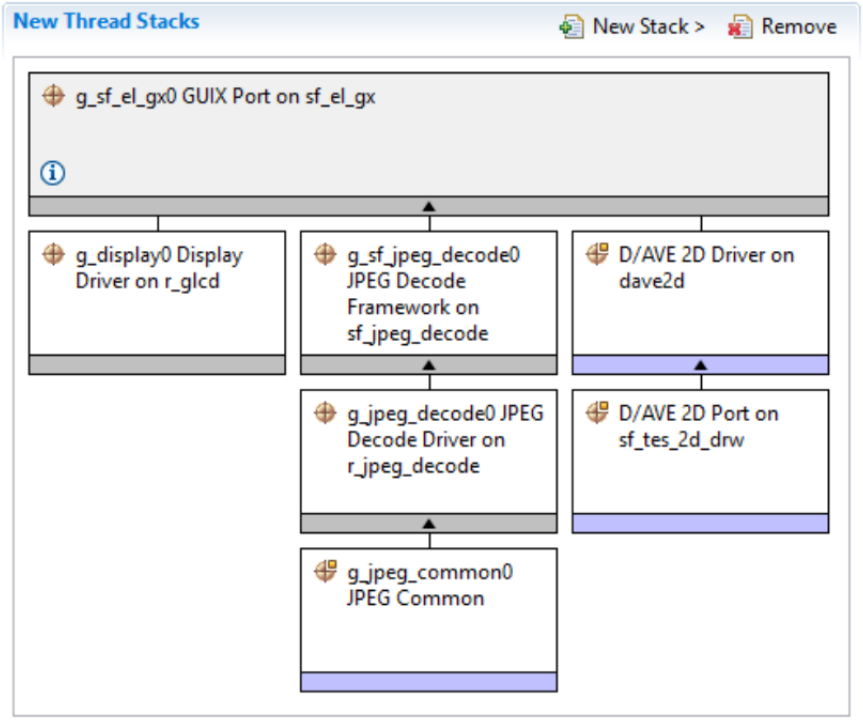


Figure 407: GUIX Synergy Port Framework Module Stack

4.3.4.5 Configuring the GUIX Synergy Port Framework Module

The GUIX Synergy Port Framework module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the GUIX Synergy Port Framework on sf_el_gx

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_el_gx0	Name of SF_EL_GX instance which will be generated by ISDE. Specify the instance name of this module. Name must be a valid C symbol.
Display Driver Configuration Instance	Inherit Graphics Screen 1, Inherit Graphics Screen 2 Default: Inherit Graphics Screen 1	Display drive configuration instance selection.
Name of User Callback function	NULL	Name of User Callback function invoked by the Module when events happen. It must be a valid C symbol and NULL is allowed.
GUIX Canvas Buffer (required only if rotation angle is FLIP or 180 degree)	Used, Not used Default: Not used	A canvas buffer must be used for FLIP or 180 degree screen rotation. The size of canvas buffer must be exactly the same as a frame buffer for the display module.
Size of JPEG Work Buffer (valid if JPEG hardware acceleration enabled)	768000	The JPEG work buffer size in bytes. Value must be a valid integer value and zero is allowed to be set if JPEG acceleration is not used. Larger buffer size shortens the drawing time. See Size of JPEG Work Buffer.

Memory section for GUIX Canvas Buffer	sdram	Name of memory section where you want to allocate the GUIX Canvas Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.
Memory section for JPEG Work Buffer	sdram	Name of memory section where you want to allocate the JPEG Work Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.
D/AVE 2 2D Frame Buffer Cache (Valid if D/AVE 2D Drawing Engine is enabled)	Enable, Disable Default: Enable	If Synergy 2D Drawing Engine (DRW) Support is enabled, the rotation is processed by Synergy DRW with texture mapping. If not enabled, the rotation is processed by software copy.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the GUIX Synergy Port Framework Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the GLCD HAL Module on r_glcd

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_display0	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.

Input - Panel clock source select	Internal clock(GLCDCLK), External clock(LCD_EXTCLK) Default: Internal clock (GLCDCLK)	Choose the panel clock source depends on your system.
Input - Graphics screen1	Used, Not used Default: Used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.
Input - Number of Graphics screen1 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen1 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen1 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen1 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.

Input - Graphics screen1 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen1 input line descending	Used, Not used Default: Not used	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen1 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.

Input - Graphics screen1 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Input - Graphics screen2	Used, Not used Default: Not used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen2 frame buffer name	fb_foreground	Custom name for frame buffer.
Input - Number of Graphics screen2 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.

Input - Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen2 input line descending	On, Off Default: Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen2 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.

Input - Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output - Horizontal total cycles	1024	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal active video cycles	800	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal back porch cycles	46	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

Output - Horizontal sync signal cycles	20	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal polarity	Low active, High active Default: Low active	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical total lines	525	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical active video lines	480	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical back porch lines	23	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal lines	10	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal polarity	Low active, High active Default: Low active	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.

Output - Format	24bits RGB888, 18bits RGB666, 16bits RGB565, 8bits serial Default: 24bits RGB888	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Endian	Little endian, Big endian Default: Little endian	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Color order	RGB, BGR Default: RGB	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Data Enable Signal Polarity	Low active, High active Default: High active	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Sync edge	Rising Edge, Falling Edge Default: Rising Edge	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Background color alpha channel	255	Specify the background color of the background screens.
Output - Background color R channel	0	Specify the background color of the background screens.
Output - Background color G channel	0	Specify the background color of the background screens.
Output - Background color B channel	0	Specify the background color of the background screens.
CLUT	Used, Not used Default: Not used	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entries consumes 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.

TCON - Hsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Vsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON1	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - DataEnable pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Panel clock division ratio	1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/12, 1/16, 1/24, 1/32 Default: 1/8	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction - Brightness	Off, On Default: Off	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction - Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Contrast	Off, On Default: Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.
Color correction - Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.

Color correction - Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Gamma correction(Red)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Green)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.

Color correction - Gamma gain G[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold G[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Blue)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain B[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold B[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).

Dithering	Off, On Default: Off	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Mode	Truncate, Round off, 2x2 Pattern Default: Truncate	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern A	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern B	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern C	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern D	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Misc - Correction Process Order	Brightness and Contrast then Gamma, Gamma then Brightness and Contrast Default: Brightness and Contrast then Gamma	Specify the color correction processing order if needed.

Line Detect Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.
Underflow 1 Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.
Underflow 2 Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	The driver needs valid interrupt priority setting and it will not work if disabled.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Decode Framework on sf_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_jpeg_decode0	The name to be used for a JPEG Decode Framework module instance.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Decode Driver on r_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.

Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Decompression interrupt priority selection.

Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Data transfer interrupt priority selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

JPEG Common Instance

ISDE Property	Value	Description
Name	g_sf_jpeg_common	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the D/AVE 2D Driver on dave2d

ISDE Property	Value	Description
No configurable parameters.		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the D/AVE 2D Port sf_tes_2d_drw

ISDE Property	Value	Description
Work memory size for display lists in bytes	32768	Work memory size for display lists selection
DRW Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	DRW INT selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

GUIX Synergy Port Framework Module Clock Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require any hardware setting except setting the ARM Cortex-M core SysTick timer.

GUIX Synergy Port Framework Module Pin Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require pin settings.

4.3.4.6 Using the GUIX Synergy Port Framework Module in an Application

These important settings are made in the Synergy Configurator and are used to initialize the module:

- Setup GLCDC configurations including the module clock setting and GLCDC interrupt priority. Typically, the configuration can be auto-generated through Synergy Configurator.
- Setup 2D Drawing engine or JPEG engine configurations including the module clock setting and hardware interrupt priorities. Typically, the configuration can be auto-generated through Synergy Configurator.

The steps in using the GUIX Port on sf_el_gx module in a typical application are:

Step 1. Initialize the SF_EL_GX control block and pass module configuration settings by calling the sf_el_gx_api_t::open API.

Step 2. Complete initialization by calling the GUIX Studio generated gx_studio_display_configure API and pass the SF_EL_GX setup function as shown in the illustration below. This function call completes the initialization of Synergy graphics hardware accelerators. Obtain the address of the root window initialized by GUIX through the call.

```
gx_studio_display_configure (MAIN_DISPLAY,
                             g_sf_el_gx0.p_api->setup,
                             LANGUAGE_ENGLISH,
                             MAIN_DISPLAY_THEME,
                             &p_window_root);
```

Step 3. Initialize the primary memory address GUIX Canvas buffer by calling the sf_el_gx_api_t::canvasInit API.

Step 4. Create the root window by calling the GUIX Studio generated gx_studio_named_widget_create API.

Step 5. Show the root screen by calling the GUIX gx_widget_show API.

Step 6. Start the GUIX system by calling the GUIX gx_system_start API.

Once GUIX system is started, the SF_EL_GX module is driven under GUIX control. The application need not execute any operations after this.

These common steps are illustrated in a typical operational flow in the following figure:

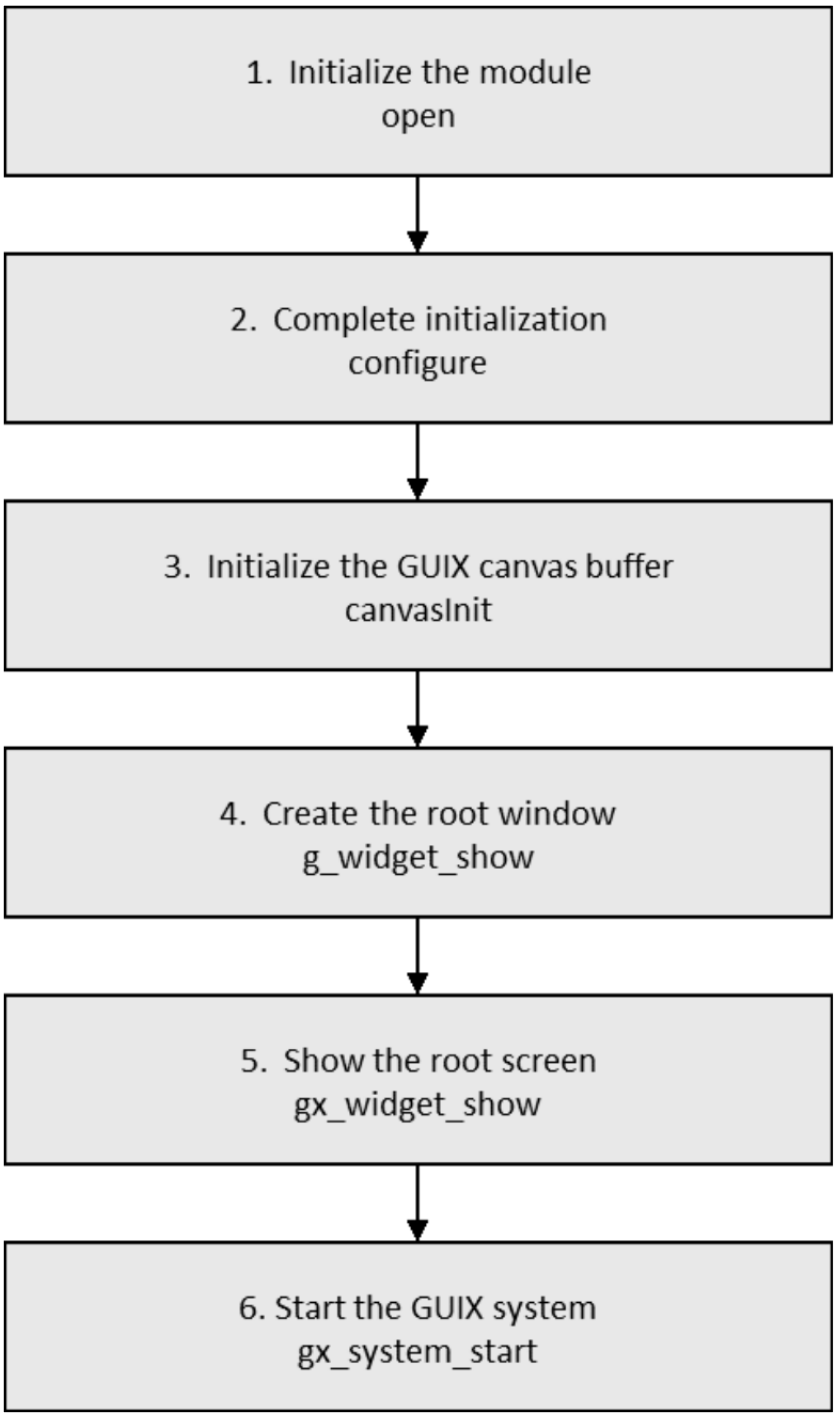


Figure 408: Flow Diagram of a Typical GUIX Synergy Port Framework Module Application

4.3.5 GUIX Source

4.3.5.1 GUIX GX_SRC Framework Introduction

The Azure RTOS GUIX Source component (GX_SRC) is the list of properties available in GUIX to modify the RTOS components of GUIX (for example, the system timer). See the GUIX User Guide Chapter 3: GUIX System Components for a list of GUIX components. The GUIX service calls have mutual exclusion for protection built in so that the application can use GUIX services while the GUIX 'engine' is managing the graphics. GUIX uses ThreadX threads, timers and message queues to manage display events and render screens.

GUIX GX_SRC Framework Module Features

The GUIX GX_SRC Framework module includes the following options:

- Disables multithread support
- Disables UTF8 Support
- Sets the system timer to match the hardware

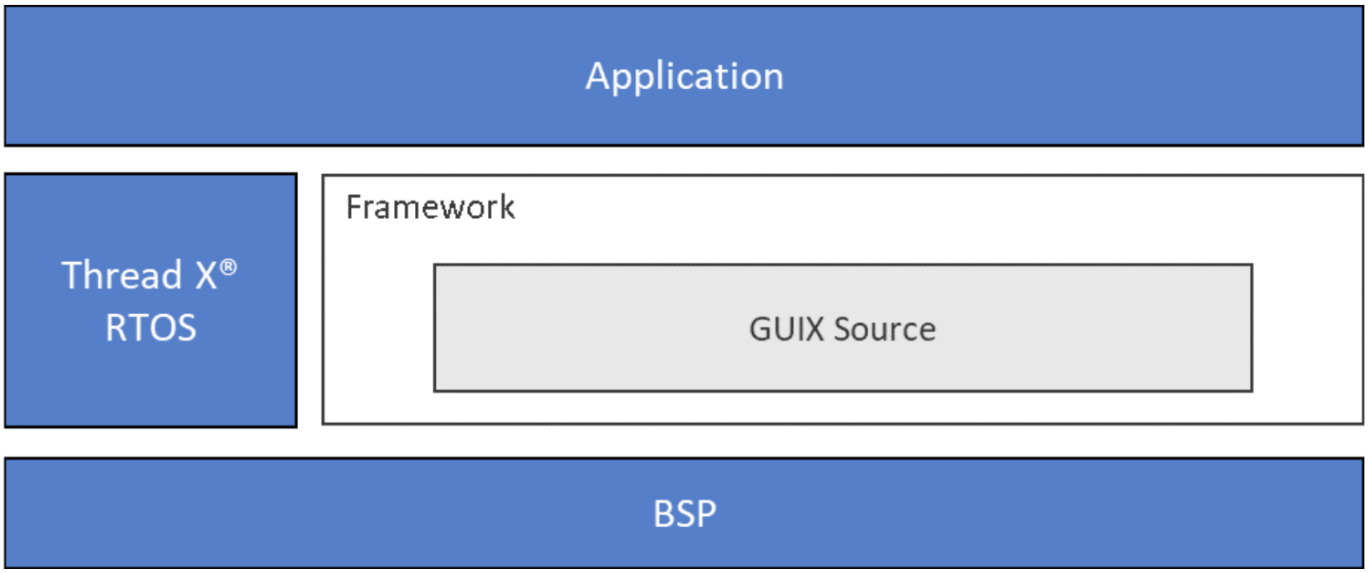


Figure 409: GUIX GX_SRC Framework Module Block Diagram

4.3.5.2 GUIX GX_SRC Framework Components Overview

The number of API functions available in GUIX in the current release is over 300 and the reader is referred to the GUIX User Guide for detailed API listings and descriptions. A list of GUIX components is discussed as follows:

GUIX Components

GUIX SYSTEM – Timers, threads and any internal objects the GUIX engine requires to run

GX_CANVAS* – Set by gx_canvas_create. In the simple GUIX design case, this function is called by the specification file from GUIX Studio so it is handled automatically.

GUIX CONTEXT – All the components that make up what GUIX is drawing into, that is, screen, canvas, brush and so on.

GX_DISPLAY* – All the components, including drawing functions that GUIX needs to draw on a canvas. GUIX display drivers are responsible for all interaction with the underlying physical screen. The display drivers have three basic functions: initialization, drawing and frame buffer display.

Initialization is responsible for preparing the physical display hardware, informing GUIX of the properties of the physical display hardware, and for informing GUIX which specific drawing functions should be used. The main display driver initialization is called from the GUIX gx_display_create API. This function is called from gx_studio_display_configure in applications that use the resource and specification.

In GUIX Studio, the specification files will set up the display driver automatically. If 16bpp is chosen in the Configure Displays dialog box in GUIX Studio, _gx_display_driver_565rgb_setup is used. If 32bpp is chosen you get _gx_display_driver_32argb_setup, and so on.

GUIX in Synergy supports the following formats:

- 32bpp (ARGB8888, RGB-888)
- 16bpp (RGB565)
- 8bpp (8bit Palette (CLUT))

The display driver function is set up in the GUIX Studio specification files if the target CPU is set to Synergy in GUIX Studio (see the IOTSG-782 module guide document for more details on setting up GUIX Studio for Synergy):

```
UINT _gx_synergy_display_driver_setup(GX_DISPLAY *display)
{
_gx_display_driver_565rgb_setup(display, GX_NULL, _gx_dave2d_buffer_toggle);
```

In the user application, the gx_studio_display_configure API configures the main display including specifying the display driver. Synergy has already tied the display driver to the SF_EL_GX instance. The SF_EL_GX driver setup function pointer supplies the display driver:

```
gx_studio_display_configure (MAIN_DISPLAY,
                             g_sf_el_gx0.p_api->setup,...
```

GX_WIDGET* – any visible thing you place on a screen; may or may not generate events

GUIX UTILITY – essentially a tool box for utility functions

- Defined as a struct data type in GUIX

4.3.5.3 GUIX GX_SRC Framework Module Operational Overview

Setting up GUIX in the Configurator

The GUIX Source component is optional. If not using the GUIX source, the prebuilt GUIX library with default settings is used. If the GUIX source component is added or any of its properties modified, the developer must click on Generate Project Content in the configurator pane and rebuild the project, or the prebuilt GuiX (gx.a) library is used with none of its properties modified.

GUIX at Startup

The following steps need to be performed before rendering any GUIX images:

- Call the gx_system_initialize API to initialize the GUIX system.
- Initialize the GUIX driver (g_sf_el_gx.p_api->open).
- Create and configure the display instance (gx_studio_display_configure called with g_sf_el_gx.p_api->setup as the driver input).
- Create and initialize the canvas for the display (g_sf_el_gx.p_api->canvasInit API).
- Prepare the display of the main screen (gx_studio_display_configure API) if using GUIX Studio specification and resource files.
- [Optionally] create any other windows or widgets if not created (gx_widget_create).
- Show the root window (gx_window_show API).
- Start the GUIX engine (gx_system_start API).

In the gx_system_initialize API, GUIX creates its own thread, event queue and timer necessary for performing all drawing, timer and GUIX related tasks. It also creates a mutex for multithreaded environments. Application threads have access to the same resources as GUIX; for example, if GUIX is doing a refresh operation, the mutex protections prevents other threads that try to access visible widgets during the refresh of a screen.

The open, setup and canvasInit API are accessible by function pointer fields of the sf_el_gx instance. One need only supply any other details except for the language and display screen name in the gx_studio_display_configure call.

GX_SRC Properties

GUIX Configurable Options for GX_SRC

ISDE property	Default value	Description
GUIX Stack Size	4096 bytes	GUIX internal thread stack size in bytes.
System Timer	20 ms	Used to calculate GX_SYSTEM_TIMER_TICKS. See explanation that follows. Equivalent option in Azure RTOS is GX_SYSTEM_TIMER_MS.
Disable Multithread Support	No (Disabled)	If not defined, ThreadX can support multiple threads by defining and using locking and unlocking functions to define critical sections. Equivalent option in Azure RTOS is GUIX_DISABLE_MULTITHREAD_SUPPORT.
Disable UTF8 Support	No (Disabled)	If defined, this disables UTF8 format string encoding in GUIX and allows only 8-bit ASCII character plus Latin-1 code page character encoding. Equivalent option in Azure RTOS is GX_UTF8_SUPPORT.

GUIX Event Queue Size	48	Size of GUIX Event Queue Size. Must be greater than zero or empty.
GUIX Thread Priority	16	Priority of GUIX Internal Thread. The value must be between 0 to 31.
GUIX Thread Time Slice	10	Time Slice value of GUIX Internal Thread. The value must be between 0(TX_NO_TIME_SLICE) to 0xFFFFFFFF.
Use User Data Field in GX_WIDGET Structure	No(default)	GUIX allows users to use gx_widget_user_data member in GX_WIDGET Structure if you say yes.
Enable partial canvas frame buffer	No (Disabled)	Enabling this option will define the GX_ENABLE_CANVAS_PARTIAL_FRAME_BUFFER to use the partial canvas frame buffer feature.
Enable horizontal canvas refresh direction	No (Disabled)	Specify the canvas refresh directions for the partial canvas frame buffer feature. By default, the canvas refreshes its dirty areas one by one. If the application requires refreshing the canvas in horizontal direction to mitigate tearing effects,this can be used.
Enable vertical canvas refresh direction	No (Disabled)	Specify the canvas refresh directions for the partial canvas frame buffer feature. By default, the canvas refreshes its dirty areas one by one. If the application requires refreshing the canvas in vertical direction to mitigate tearing effects,this option can be used.
Disable arc drawing support	No (Disabled)	If defined, removes support for the arc-drawing functions circle, arc, pie, and ellipse.
Disable software decoder support	No (Disabled)	If defined, reduce the GUIX library footprint when runtime decode of jpg or png files is not required.
Disable binary resource support	No (Disabled)	If defined, reduce your GUIX library footprint when only C source code format resource files are used.

Repeat button initial tics	10	This value defines how long the button waits before beginning to send repeated GX_EVENT_CLICKED events.
Maximum number of unique dirty list entries	64	This value defines the maximum number of unique dirty list entries that can be maintained by one canvas.
Maximum nesting of context	8	This value defines the maximum nesting of the drawing context stack.
Maximum input capture nesting	4	This value defines size of the stack used to maintain the list of widgets that have captures the user input.
Cursor blink interval	20	This value defines the rate at which the input cursor blinks for text input widgets.
Multi line index cache size	32	This value defines the size of the list-start index cache maintained by the multi-line text view and multi-line text input widgets.
Multi line text button maximum lines	4	This value determines the number of text pointers needed by the worst case multi-line text button.
Maximum number of polygon edges	10	This value determines the most complex polygon that can be drawn by GUIX.
Numeric scroll wheel string buffer size	16	This value determines the maximum length of the string required to display the assigned integer values.
Circular gauge animation delay	5	This value defines the number of GUIX timer ticks (50 ms) between updates of a circular gauge configured to animate the needle movement between last and current angular position.
Numeric prompt buffer size	16	This value defines the size of buffer allocated to convert an integer value assigned to the prompt to an ascii string.

Animation pool size	6	This value defines the size of animation pool from which animation information structures can be dynamically allocated and returned.
Mouse support	Disabled	Enables the mouse support. This definition should only be enabled when a mouse (not a touch screen) must be supported.
Hardware mouse support	Disabled	When enabled, the GUIX display driver utilizes hardware mouse cursor drawing support.
Font kerning support	Disabled	Enables font kerning support.
Maximum string length	102400	This value defines the maximum length of a string, which is used to test invalid strings.
Disable brush alpha support	No (Disabled)	Allows brush alpha support to be disabled. This helps eliminate runtime overhead and library footprint increase for drawing non-arc graphics, pixelmaps, and fonts with an alpha value defined by the drawing context brush while running at 16 bpp and higher color depths.
Maximum string length	102400	This value defines the maximum length of a string, which is used to test invalid strings.

GUIX Options Available in the Azure RTOS User Guide (Non-configurable in ISDE)

MACRO	Default value	Description
GX_SYSTEM_TIMER_TICKS	2 ticks (20 ms)	GUIX timer frequency (interval on which the timer thread entry function checks for tasks (for example, periodic tasks, timeouts) that need to be executed.
GX_TICKS_SECOND	2 ticks	This is only used if porting GUIX to another RTOS other than ThreadX.
GX_MAX_VIEWS	32	Number of simultaneous views.

GX_MAX_ACTIVE_TIMERS	32	Number of timers available in GUIX to be assigned to one or more widgets.
GX_DISABLE_THREADX_BINDING	Not enabled	If not defined, GUIX expects a ThreadX RTOS; GUIX can work with other RTOSs, but this is not supported in Synergy.
GX_DISABLE_THREADX_TIMER_SOURCE	Not enabled	If defined, disables ThreadX timer source to use a different timer source.

Disabling Multithread Support

The gx_system_initialize API creates the GUIX drawing and event processing thread. That thread is started when the gx_system_start API is called. GUIX application threads may wish to invoke the GUIX API while the GUIX system thread is running. An application thread could, for example, create a pop-up error window and display it by attaching it to the root window. The problem is that two or more threads accessing internal GUIX resources simultaneously will cause undefined results. Therefore, in a multithreaded environment, GUIX must protect critical code sections where it updates linked lists and other internal objects. To do so, GUIX uses mutexes to guard critical code sections. If Multithread Support property is not disabled, GUIX macros enable protection of the critical sections. It is generally recommended that if a system is not severely constrained by resources (memory), that this option be left disabled.

If the application does not call any GUIX API, then the only thread affecting GUIX is the system thread; disabling multithreaded support will reduce the overhead of protecting critical code sections and improve performance.

GUIX System Timer

The GUIX System timer is the interval on which the GUIX timer executes and is used by GUIX for periodic processing needs inside GUIX. The system timer It should be a multiple of the ThreadX timer, which is typically defined to be 10 ms (100 ticks/sec). Example:

GUIX System Timer is to 10

ThreadX timer is 100

GUIX timer tick = ((10 ms * 100 ticks per second) / 1000 ms per second) == 1 tick

This sets the GUIX timer tick to one tick. If the GUIX System timer is not a multiple of the ThreadX timer, it will be rounded down. In the example above, if it were set to 17, the GUIX timer tick would still be "1". To make the GUIX timer tick every other ThreadX timer tick, set the System timer property in GX_SRC to 20;

GUIX timer tick = ((20 * 100) / 1000) == 2.

UTF8 Support

By default, UTF8 is enabled. If UTF8 is disabled, GUIX is limited to ASCII type (one byte per glyph). There is always runtime overhead associated with UTF8 support even if the application is only using ASCII. This is because GUIX has to call a function to compute the next character value in the string when using UTF8 because it cannot assume it is a one-byte character.

In lieu of disabling UTF8, the application can avoid using character values higher than 255. When UTF8 is disabled, this tells GUIX that each character is simply one byte and it does not need to call the next-character-compute function.

Display Memory Architecture

There are several display memory architectures supported by GUIX. The display memory architecture is really defined by the display driver, which is a very small hardware porting layer, and does not affect the core GUIX library code. The most common model is to provide two canvas memory areas: one the "working" buffer and one the "visible" buffer. GUIX executes drawing updates to the working canvas, then toggles the working and visible canvas buffers when a drawing sequence is completed.

Four basic memory models are described as follows:

Models 1 and 2 contain only GRAM and Frame Buffer respectively in the following diagram, and would be recommended for only very small displays as it is slow. In Model 1, the display provides its own memory large enough to hold the pixel data. This external memory is usually called "GRAM" or Graphics Ram, and some non-random access such as SPI access is provided to the core CPU. In this model, the display driver executes all drawing operations by writing pixel data over the SPI interface.

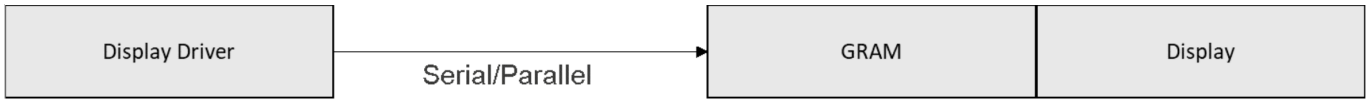


Figure 410: Model 1

Figure 411: Model 2

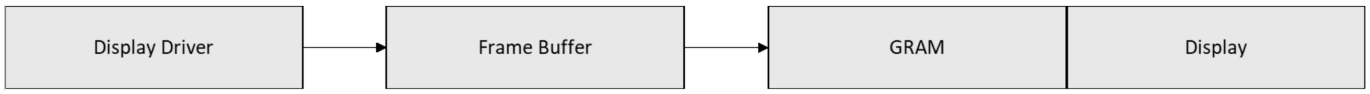


Figure 412: Model 3 (Local Frame Buffer + External GRAM)

Model 3 is a combination of the first two models. In this model, sufficient local memory exists to hold one frame buffer. In addition, the display device provides an external GRAM and automatically refreshes itself using the data provided in the GRAM. This architecture benefits from improved update efficiency because the user can transfer the modified portion of the local frame buffer to the external GRAM in one block transfer, often utilizing on-board DMA channels. This model also eliminates the tearing and flicker that can be present in either of the first two models, because only completed graphics content is copied to the external GRAM.

Model 4 requires that sufficient memory is present to provide two local frame buffers. In this case, GUIX treats one frame buffer as the active frame buffer, and the other as the working frame buffer. When a display update or drawing operation is in progress, it takes place in the working buffer. When the drawing operation completes, the buffers are toggled, and the working buffer becomes the active buffer and the active buffer becomes the working buffer. Like Model 3, this model also eliminates screen flicker and tearing that can be observed in a single buffered system.

Model 4 Ping Pong frame buffers:

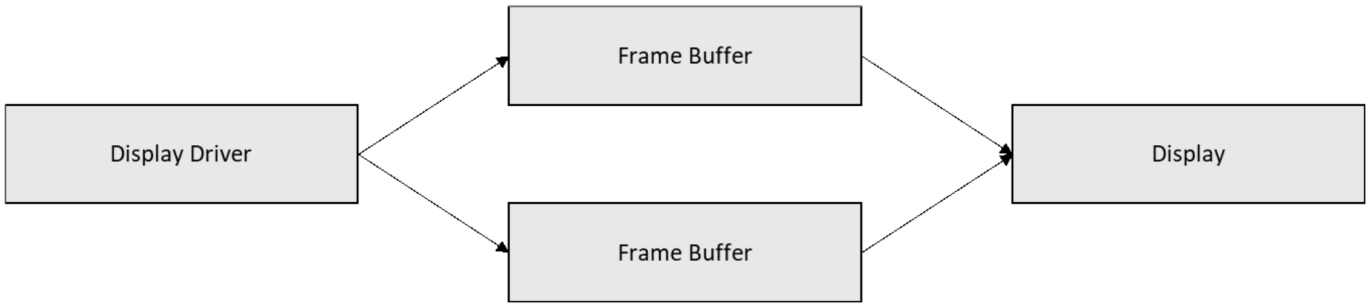


Figure 413: Model 4 (Ping-Pong Frame Buffers)

Synergy (SF_EL_GX module) implements Model 4 (although Model 3 is also possible).

Receiving Events from the Hardware

Signals in GUIX

Signals are used to get information such as hardware events up to the GUIX level. Hardware events are transduced by the hardware and message framework drivers into GX_SIGNALs by the time they reach the GUIX event queue. The GX_SIGNAL macro is a macro that combines the ID of the widget (user defined in GUIX Studio) that generated an event with the event type the widget has generated into a unique gx_event_type identifier code. For more details about GX_EVENT types and signals, please see the GUIX™ Synergy Port Framework Module Guide.

Drivers for the Hardware Interface

In the Synergy driver hardware, a ThreadX periodic thread task is created that continually checks for hardware events. When an event is detected, the hardware driver transmits information about this event to the thread task. The Touch Panel V2 Framework module scans data from a touch controller and invokes the user-registered touch panel callback when a touch event occurs. If the user callback is not registered, the sf_touch_panel_v2_api_t::touchDataGet API function can be used to retrieve the data. In the example of the touchscreen, the touch panel driver would relay X and Y coordinates, and event type.

```
/* User callback*/
typedef struct st_sf_touchpanel_v2_callback_args
{
    sf_touch_panel_v2_payload_t     payload;       // Touch data and event provided
to the user during callback
    void const                  * p_context;      // Context provided to user during
callback
} sf_touchpanel_v2_callback_args_t;
/* Touch data payload */
typedef struct st_sf_touch_panel_v2_payload
```

```
{
    int16_t                         x;              // X coordinate.
    int16_t                         y;              // Y coordinate.
    sf_touch_panel_v2_event_t       event_type;     // Touch event type.
} sf_touch_panel_v2_payload_t;
```

The user callback must be registered to obtain touch data. The Touch Panel V2 Framework module scans data from a touch controller and invokes the user-registered callback. If the user callback is not registered, the sf_touch_panel_v2_api_t::touchDataGet API function can be used to retrieve the data. If the user callback is registered, then the framework will invoke the callback when touch events occur.

Example: (User-defined touch panel callback function)

```
void g_sf_touch_panel_cb(sf_touchpanel_callback_args_t * p_args)
{
    bool send_event = true;
        GX_EVENT gxe;
        switch (p_args->payload.event_type)
        {
        case SF_TOUCH_PANEL_V2_EVENT_DOWN:
            gxe.gx_event_type = GX_EVENT_PEN_DOWN;
            break;
        case SF_TOUCH_PANEL_V2_EVENT_UP:
            gxe.gx_event_type = GX_EVENT_PEN_UP;
            break;
        case SF_TOUCH_PANEL_V2_EVENT_HOLD:
        case SF_TOUCH_PANEL_V2_EVENT_MOVE:
            gxe.gx_event_type = GX_EVENT_PEN_DRAG;
            break;
        case SF_TOUCH_PANEL_V2_EVENT_INVALID:
            send_event = false;
            break;
        default:
            break;
        }
```

```
    if (send_event)
    {
        /* Send event to GUIX */
        gxe.gx_event_sender         = GX_ID_NONE;
        gxe.gx_event_target         = 0;  /* the event to be routed to the widget
that has input focus */
        gxe.gx_event_display_handle = 0;
        gxe.gx_event_payload.gx_event_pointdata.gx_point_x = p_args->payload.x;
  #if defined(BSP_BOARD_S7G2_SK)
        gxe.gx_event_payload.gx_event_pointdata.gx_point_y = (GX_VALUE)(320 -
p_payload->y);
  #else
        gxe.gx_event_payload.gx_event_pointdata.gx_point_y = p_args->payload.y;
  #endif
        gx_system_event_send(&gxe);
    }
}
```

The user callback function can update the GX_EVENT data type and fill in the following fields as needed:

```
     /* Send event to GUIX */
        gxe.gx_event_sender         = GX_ID_NONE;
        gxe.gx_event_target         = 0;  /* the event to be routed to the widget
that has input focus */
        gxe.gx_event_display_handle = 0;
        gxe.gx_event_payload.gx_event_pointdata.gx_point_x = p_args->payload.x;
  #if defined(BSP_BOARD_S7G2_SK)
        gxe.gx_event_payload.gx_event_pointdata.gx_point_y = (GX_VALUE)(320 -
p_payload->y);
  #else
        gxe.gx_event_payload.gx_event_pointdata.gx_point_y = p_args->payload.y;
  #endif
        gx_system_event_send(&gxe);
```

SF_TOUCH_PANEL_V2_EVENT_DOWN, SF_TOUCH_PANEL_V2_EVENT_UP, and other touchscreen panel events are defined in the touchscreen header files. The touchscreen driver also defines its only payload field, since it needs to communicate some value(s) to the application besides event type. For a touchscreen event, the touchscreen payload includes an x and y coordinate. The GX_EVENT contains a union of field for payload. In this case, the GX_EVENT gx_event_payload field uses the GX_POINT gx_event_pointdata which holds two values for x and y coordinates.

Now, the application can send this event over to the GUIX event queue to let GUIX handle it by calling gx_system_event_send API.

GUIX periodically checks the event queue, and sees the event just added to the queue. In the case of the touchscreen panel, it uses the x,y coordinates to find the window widget currently attached to the root window, and searches all the child widgets contained by that window for the highest one (Z order) that contains this coordinate. Then GUIX typically defers to the child widget's parent event handler for passing off further handling of the event.

Screen Management

Root and Child Windows

Each visible canvas has a root window. The root window is the container for all windows and widgets of the display.

Attaching and Detaching Screens

If the new screen is a child of the parent screen, the application must attach the new screen and detach the old screen using gx_widget_attach and gx_widget_detach (order is not important). If moving back to a parent screen, the application does not attach the parent screen but calls gx_window_show on the parent screen. A simple algorithm for swapping out screens follows:

```
void ToggleScreen(GX_WIDGET *new_win, GX_WIDGET *old_win)
{
    if (!new_win->gx_widget_parent)
    {
        gx_widget_attach(root, new_win);
    }
    else
    {
        gx_widget_show(new_win);
    }
    gx_widget_hide(old_win);
}
```

In the following code, GUIX has detected a button widget clicked in window1, for switching from

window1 to window2. It attaches the new window (window2) to the root window, and detaches the old window (window1):

```
UINT window1_handler(GX_WINDOW *widget, GX_EVENT *event_ptr)
{
    gx_window_event_process(widget, event_ptr);
    switch (event_ptr->gx_event_type)
    {
        case GX_SIGNAL(ID_WINDOWCHANGER, GX_EVENT_CLICKED):
        if(button_enabled)
        {
            show_window((GX_WINDOW*)&window2, (GX_WIDGET*)widget, true);
        }
        break;
       default:
        gx_window_event_process(widget, event_ptr);
        break;
    }
    return result;
}
static UINT show_window(GX_WINDOW * p_new, GX_WIDGET * p_widget, bool detach_old)
{
UINT result = gx_window_event_process(widget, event_ptr);
switch (event_ptr->gx_event_type)
{
   case GX_SIGNAL(ID_WINDOWCHANGER, GX_EVENT_CLICKED):
      if (!p_new->gx_widget_parent)
{
err = gx_widget_attach(p_window_root, p_new);
}
else
{
err = gx_widget_show(p_new);
}
gx_system_focus_claim(p_new);
```

```
GX_WIDGET * p_old = p_widget;
if (p_old && detach_old)
{
if (p_old != (GX_WIDGET*)p_new)
{
gx_widget_detach(p_old);
}
}
break;
   default:
gx_window_event_process(widget, event_ptr);
break;
}
return result;
}
```

Both methods are perfectly acceptable ways to move between screens, with the latter having more complexity. It does not matter which order is followed in attaching or detaching screens. GUIX just marks the to-be visible window as dirty and re-draws that window after the application is finished modifying the tree* of visible widgets. In other words, GUIX does not actually draw the window as part of the "attach" API call.

Note

GUIX maintains a tree structured lists of visible objects for linking child widgets (many) to their parent widget (one). When a window is swapped out with another, the tree is modified to include the new (visible) window and remove the old (soon to be not visible) window.

Attaching and Detaching Widgets

GUIX keeps a linked list of child widgets (a window is a widget type) that belong to or are attached to a parent. Child windows are attached to the root window. When a widget is attached, it is added to the end of the list. There is a first and last child marker of this list (not a circular list). Before a child widget can be attached to a parent widget, it must not be a child of another parent. If so, it is detached from that parent first.

When it is detached, it is removed from the list. However, it is not deleted, so it is still available to be reattached without needing to be recreated. In the meantime, it cannot draw, and it has no parent, but it can still receive events and the control block for that widget is still valid.

Detaching vs Deleting Widgets

When a widget is detached, it is removed from the active list of widgets maintained by GUIX. However, it is not deleted, so it is still available to be reattached without needing to be recreated. In the meantime, it cannot draw, and it has no parent, but it can still receive events and the control

block for that widget is still valid.

The gx_widget_delete API first detaches the widget if it still has a parent, then, it clears out the control block. If the control block was dynamically allocated, the memory for the control block is freed. If the gx_widget_delete API was applied to a widget, the widget will need to be re-created before it can be displayed again.

When to Create Widgets

GUIX Studio enables a developer to create screens, name widgets, set event and draw callback functions much more easily than creating them by source code in GUIX APIs. In GUIX Studio, the developer can choose which widgets are created at start up time (default setting), or it can enable runtime creation by checking the Run Time Allocation property of a widget. If this is enabled, the application must create the widget when it needs it. Otherwise, all the 'static' widgets are created at start up. This requires that there be enough memory for these 'static' widgets. If there is, there really is no reason to not create all the widgets at start time. However, in an application with a large number of screens, or limited memory, it may be necessary to create widgets as needed at runtime.

To designate a widget for dynamic memory allocation, check the box for Runtime Allocate in GUIX Studio. In the specification files created by GUIX Studio, the 'style flag' of the widget (and any children of the widget) sets the GX_ STYLE_DYNAMICALLY_ALLOCATED attribute. When the application creates the widgets, it will allocate memory using the memory allocator specified in gx_system_memory_allocator_set. Note that this requires that the application set up a memory pool before starting the GUIX services. The project for this module guide demonstrates this.

Drawing at Run Time

Some screens require drawing at run time inevitably. These would be line draw and text writing API for updating information on a screen in real time. Dynamic string creation is accomplished using sprintf and gx_utility_ltoa functions; these are useful for printing run time values in the string. Some of the APIs used for that screen for drawing are listed below as examples:

```
UINT _gx_canvas_line_draw(GX_VALUE x_start, GX_VALUE y_start, GX_VALUE x_end,
GX_VALUE y_end);
UINT _gx_canvas_text_draw(GX_VALUE x_start, GX_VALUE y_start, GX_CONST GX_CHAR
*string, INT length);
```

Normally, drawing to a screen is done by a method called deferred drawing, where GUIX internally manages when a window or part of a window should be redrawn so as to improve drawing efficiency.

However, when an application wants immediate drawing to a canvas, it must call the gx_canvas_drawing_initiate API before drawing anything. When completed, the application calls gx_canvas_drawing_complete to signal GUIX to let deferred drawing resume.

Run time drawing requiring memory allocation and deallocation is demonstrated in the rotation of the thermostat in the gux_gx_src_mg_ap.c file in the module guide project.

Note that in SSP 1.3.0/GUIX 5.3.3, to 'rotate' or otherwise redraw an image at run time the pixelmap data must not be compressed. If you anticipate needing to redraw an image differently, such as using rotation, make sure to uncheck the **Compress Output** for the **pixelmap** resource in GUIX

Studio (v5.3.3.7 and later). To edit a pixelmap resource, open the **Pixelmaps** bar by clicking on the (+**) icon -> click on **Custom** -> double click on the graphic item -> **Uncheck Compress Output** (it is enabled by default).

Memory Allocation in GUIX Applications

Allocating Memory

Dynamic memory allocation is supported (for example, heap memory) in GUIX. The gx_system_memory_allocator_set API lets the application assign a memory allocation and a memory free service. This API is called at program startup, after gx_system_initialize, and before any GUIX services that require dynamic memory allocation.

GUIX services that require a runtime memory allocation and de-allocation service include:

- Loading binary resources from external storage into the GUIX runtime environment.
- The software runtime jpeg image decoder.
- The software runtime png image decoder.
- Using text widgets with GX_STYLE_TEXT_COPY.
- Runtime pixelmap resize and rotation utility functions.
- Runtime screen and widget control block allocation.

Dynamic memory allocation is required for pixelmap rotation in the project for this module guide.

Dynamic allocation lets an application import resources like fonts, language, and images at runtime from non-volatile memory like a flash drive or a URL source. Most GUIX applications are small enough that this is not required. GUIX resources can be loaded and statically linked at compile time using the pixelmaps created in GUIX studio, which decode the resource from jpeg to pixelmap format.

Widget Creation: Static vs Runtime

If you have plenty of memory and your control blocks are all statically allocated, for example, the **Runtime Allocated** property for the widget in GUIX Studio is not checked, then there is no reason not to create them during program startup. Creating your GUIX screens is just setting their initial parameters using the data written to the specifications file. You can always modify any of these parameters at any time after the screen has been created.

If you want your application to be more RAM efficient, set the **Runtime Allocated** property for some or all of your screen widgets in GUIX Studio. In your application, set up a memory pool to use for dynamically allocating your control blocks. This is done by calling the gx_system_memory_allocator_set API, which takes as its input a memory allocate function pointer and a memory release function pointer. These user defined functions will typically use the ThreadX tx_byte_allocate and tx_byte_release API, respectively, but other memory allocation services can be used.

Dynamic widget allocation is generally handled in the GUIX Studio specification file. In the specification file, gx_studio_widget_create will dynamically allocate those widget control blocks assigned for dynamic allocation. In the following example, the medical screen widget is designated for dynamic allocation as indicated by the style symbol: GX_STYLE_DYNAMICALLY_ALLOCATED:

```
GX_CONST GX_STUDIO_WIDGET meds_screen_define =

{
```

```
    "meds_screen",
    GX_TYPE_TEMPLATE,   /* widget type                 */
    ID_MEDS_SCREEN,   /* widget id                    */
    GX_STYLE_BORDER_THIN|GX_STYLE_DYNAMICALLY_ALLOCATED,   /* style flags   */
    GX_STATUS_ACCEPTS_FOCUS,   /* status flags             */
```

Then widget create handler in the specification file will check if a widget should be allocated dynamically:

```
static GX_WIDGET *gx_studio_nested_widget_create(GX_BYTE *control,
GX_CONST GX_STUDIO_WIDGET *definition, GX_WIDGET *parent)
{
    UINT status = GX_SUCCESS;
    GX_WIDGET *widget = GX_NULL;
    while(definition && status == GX_SUCCESS)
    {
                if (definition->style & GX_STYLE_DYNAMICALLY_ALLOCATED)
                {
                    status = gx_widget_allocate(&widget,
definition -> control_block_size);
                    ...
```

Note that the application can call gx_widget_allocate at runtime as well.

When gx_widget_delete is called on this widget, GUIX will call the memory free function specified by the application to release the control block memory. Applications that have hundreds of screens and limited RAM would almost always be set up this way.

When a widget is dynamically allocated, the control blocks for any child widget automatically become dynamically allocated. You cannot and are not required to set this flag for child widgets. Once you set the flag at any level in e^2 studio, all children of the dynamically allocated parent also become dynamically allocated.

Multiple Canvases and Layers

GUIX in Synergy is limited to one 'simple' canvas. A simple canvas is an off-screen drawing area used by the application. GUIX does nothing directly with a simple canvas, but the application can use a simple canvas to render complex drawing to an off-screen buffer, and then use this off-screen buffer to refresh the visible canvas when needed. Synergy does not currently support multiple canvases. Most GUIX applications are simple enough that multiple canvases not required. The benefit of using multiple canvases is being able to assign one or more canvases to fast memory to

meet performance requirements. Multiple canvases can also be used for special effects such as fading in and fading out.

Support for multiple graphics layers (multiple overlaid frame buffers) is also not supported.

Timers

The gx_system_timer start API is called on a widget after GUIX services are initialized and running. The input includes a timer ID for this timer (zero is not allowed) defined by the application. To stop the timer, the gx_system_timer_stop API is called on the same widget. If a non-zero timer ID is provided, GUIX will search for all timers attached to this widget for one matching that ID. If a zero timer ID is provided, GUIX will stop and detach all timers for the specified widget.

On starting a timer, an available timer is allocated from GUIX's free list of timers and added to the list of active timers. When a timer is stopped it is moved back from the active list of timers to the free list.

When a timer expires, GUIX sends the GX _EVENT_TIMEOUT for that widget (timer owner). The event handler for that widget should handle the timeout event and perform any necessary tasks. A timer will be reset up to the number of times specified in the gx_system_timer_start API.

GUIX uses ThreadX timer for certain visual effects as well as time out expiration events. Fading out and sprite animation are two examples.

Versions in GUIX and GUIX Studio

GUIX library and Studio versions work as follows. The GUIX library has major.minor.service_pack version information. So it is always 5.3.3 or 5.4.0, three fields. The Studio version number is <GUIX_LIB_VERSION>.Studio Revision. Do not forget that Studio requires the GUIX library as part of the Studio build, this is how we render the GUIX widgets within the Studio target view panel. So the first Studio release based on 5.3.3 library is 5.3.3.0. The latest Studio release based on 5.3.3 library is 5.3.3.7. The first Studio release based on upcoming 5.4.0 library will be 5.4.0.0. So Studio release adds one field to GUIX library release.

Porting GUIX projects from SSP 1.2.0 and 1.2.1 to SSP 1.3.0

- To port GUIX projects previously built using SSP 1.2.0 or 1.2.1, follow these steps:
    1. Install GUIX Studio 5.3.3.7.
    2. Open the gxp project file used for the GUIX project in 1.2.0/1.2.1 in GUIX Studio 5.3.3.7.
    3. Update the GUIX Library Version in the **Configure Display** screen (right click on the main display window) and choose 5.3.3.
    4. Generate the new specification and resource files and copy them to your project file if they do not already get written to your src folder or subfolder automatically.
    5. In the Synergy configurator, choose **SSP 1.3.0**, and click on **Generate Project Content**.
    6. Build the project.
- If compile errors result referring to dlist_start and dlist_indirect not being defined, delete the tes and sf_tes_2d_draw folder in the synergy\ssp\src\framework folder. Then regenerate project files by clicking on **Generate Project Content** and the **Build project** icon.
- Compile errors result referring to the following undefined GUIX functions may result if the wrong version of GUIX is set in GUIX Studio:
    1. _gx_dave2d_glyph_8bit_draw
    2. _gx_dave2d_glyph_4bit_draw
    3. _gx_dave2d_glyph_1bit_draw

Check the headers in the specification and resource files to verify which version of GUIX Studio was used. You can check function headers in any GUIX source file (assuming you have access to it) to verify the GUIX version is 5.3.3.

GUIX GX_SRC Framework Module Important Operational Notes and Limitations

GUIX GX_SRC Framework Module Operational Notes

GUIX GX_SRC Framework Module Limitations

- GUIX in Synergy does not support 4bpp (grayscale) or 1bpp (monochrome)
- SF_EL_GX does not support a system with more than two frame buffers.
- SF_EL_GX supports only one GUIX canvas system.
- SF_EL_GX makes use of only one graphics layer in the DISPAY module.
- Refer to the most recent SSP Release notes for additional limitations when using this module.

4.3.5.4 Including the GUIX GX_SRC Framework Module in an Application

This section describes how to include the GUIX GX_SRC Framework module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the GUIX GX_SRC Framework module to an application, click on the Add GUIX Source block below the *GUIX on gx* component in the Thread Stack Panel and choose New.

GUIX GX_SRC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
GUIX Source	Threads	**New Stack**> **X-Ware**> **GUIX**> **GUIX Source**

When the GUIX GX_SRC Framework module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

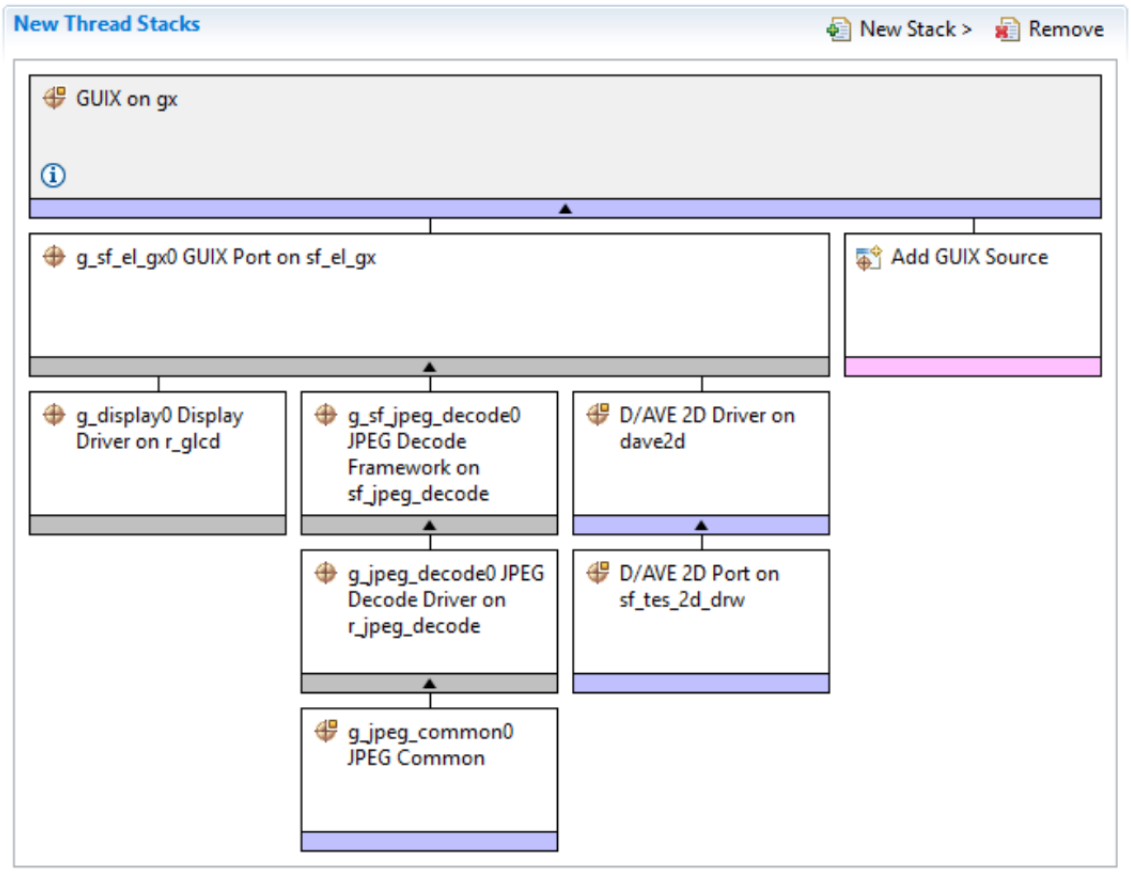


Figure 414: GUIX GX_SRC Framework Module Stack

4.3.5.5 Configuring the GUIX GX_SRC Framework Module

The GUIX GX_SRC Framework module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the GUIX GX_SRC Framework

ISDE Property	Value	Description
GUIX Stack Size		GUIX internal thread stack size in bytes. Must be greater than zero or empty.

GUIX System Timer (Milliseconds)		GUIX system timer. Must be a multiple of TX_TIMER_TICKS_PER_SECOND or empty.
GUIX Timer Rate		GUIX timer rate as a multiple of the ThreadX tick interrupt rate. Must be greater than zero or empty.
Disable Multithread Support	Yes, No Default: No	If your application has only one thread which utilizes the GUIX API services, say yes to reduce system overhead.
Disable UTF8 Support	Yes, No Default: No	GUIX disables UTF8 support if you say yes.
GUIX Event Queue Size		Size of GUIX Event Queue Size. Must be greater than zero or empty.
GUIX Thread Priority	16	Priority of GUIX Internal Thread. The value must be between 0 to 31.
GUIX Thread Time Slice	10	Time Slice value of GUIX Internal Thread. The value must be between 0(TX_NO_TIME_SLICE) to 0xFFFFFFFF.
Use User Data Field in GX_WIDGET Structure	Yes, No Default: No	GUIX allows users to use gx_widget_user_data member in GX_WIDGET Structure if you say yes.
Enable partial canvas frame buffer	Yes, No Default: No	Enabling this option will define the GX_ENABLE_CANVAS_PARTIAL_FRAME_BUFFER to use the partial canvas frame buffer feature.
Enable horizontal canvas refresh direction	Yes, No Default: No	Specify the canvas refresh directions for the partial canvas frame buffer feature. By default, the canvas refreshes its dirty areas one by one. If the application requires refreshing the canvas in horizontal direction to mitigate tearing effects,this can be used.

Enable vertical canvas refresh direction	Yes, No Default: No	Specify the canvas refresh directions for the partial canvas frame buffer feature. By default, the canvas refreshes its dirty areas one by one. If the application requires refreshing the canvas in vertical direction to mitigate tearing effects,this option can be used.
Disable arc drawing support	Yes, No Default: No	Reduce the GUIX library code size and GX_DISPLAY structure size by removing support for the arc-drawing functions circle, arc, pie, and ellipse.
Disable software decoder support	Yes, No Default: No	If your application does not require runtime decode of jpg or png files, select yes to reduce the GUIX library footprint.
Disable binary resource support	Yes, No Default: No	If your application is using only C source code format resource files, select yes to reduce your GUIX library footprint.
Repeat button initial tics		This value defines how long the button waits before beginning to send repeated GX_EVENT_CLICKED events.
Maximum number of unique dirty list entries		This value defines the maximum number of unique dirty list entries that can be maintained by one canvas.
Maximum nesting of context		This value defines the maximum nesting of the drawing context stack.
Maximum input capture nesting		This value defines size of the stack used to maintain the list of widgets that have captures the user input.
Cursor blink interval		This value defines the rate at which the input cursor blinks for text input widgets.
Multi line index cache size		This value defines the size of the list-start index cache maintained by the multi-line text view and multi-line text input widgets.

Multi line text button maximum lines		This value determines the number of text pointers needed by the worst case multi-line text button.
Maximum number of polygon edges		This value determines the most complex polygon that can be drawn by GUIX.
Numeric scroll wheel string buffer size		This value determines the maximum length of the string required to display the assigned integer values.
Circular gauge animation delay		This value defines the number of GUIX timer ticks (50 ms) between updates of a circular gauge configured to animate the needle movement between last and current angular position.
Numeric prompt buffer size		This value defines the size of buffer allocated to convert an integer value assigned to the prompt to an ascii string.
Animation pool size		This value defines the size of animation pool from which animation information structures can be dynamically allocated and returned.
Mouse support	Enabled, Disabled Default: Disabled	Enables the mouse support. This definition should only be enabled when a mouse (not a touch screen) must be supported.
Hardware mouse support	Enabled, Disabled Default: Disabled	When enabled, the GUIX display driver utilizes hardware mouse cursor drawing support.
Font kerning support	Enabled, Disabled Default: Disabled	Enables font kerning support.
Maximum string length		This value defines the maximum length of a string, which is used to test invalid strings.

Disable brush alpha support	Yes, No Default: No	When running at 16 bpp and higher color depths, GUIX optionally supports drawing non-arc graphics, pixelmaps, and fonts with an alpha value defined by the drawing context brush. Supporting this drawing mode introduces a small runtime overhead and library footprint increase, which can be eliminated by defining this flag if the application do not require alpha-blending drawing support.
Show linkage warning	Enabled, Disabled Default: Enabled	Select whether or not to show linkage warning.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

GUIX GX_SRC Framework Module Clock Configuration

The GUIX GX_SRC Module is a logical module and therefore does not require any hardware setting except setting the ARM Cortex-M core SysTick timer.

GUIX GX_SRC Framework Module Pin Configuration

The GUIX GX_SRC Module is a logical module and therefore does not require pin settings.

4.3.5.6 Using the GUIX GX_SRC Framework Module in an Application

The typical steps in using the GUIX Framework module in an application are (assuming the instance of the GUIX driver is g_sf_el_gx and the instance of the SPI driver for the LCD is g_rspi_lcdc):

1. Create a memory pool for dynamic allocation in GUIX.
2. Initialize GUIX with the gx_system_initialize function.
3. Set the memory rotate and memory free services for GUIX to use using the gx_system_memory_allocator_set API
4. Initialize GUIX drivers using the open API (g_sf_el_gx.p_api->open)
5. Create and initialize the main display usingthe gx_studio_display_configure API. This uses the open function of the SF_EL_GX driver instance as one of the input parameters.
6. Initialize the memory address of the canvas with the canvasInit API for the SF_EL_GX driver
7. Loop through all the widgets defined in the GUIX studio resource files. Create the root window and each widget using the gx_studio_named_widget_create API.
8. Show the root window using the gx_widget_show API.
9. Start the GUIX system with the gx_system_start API.
10. Open the SPI driver to initialize the LCD using the open API for the SF_EL_GX driver.
11. Set up the LCD display with the ILI9341V_Init function.

The next steps describe the invoking of the touch panel user callback function when a touch event occurs and use GUIX services to process them and update the image display:

1. Register a touch panel user callback in the application code to receive touch data. (Get the data using the sf_touch_panel_v2_api_t::touchDataGet API function if the user callback is not registered.)
2. Operate on the received touch data as needed.
3. Send the event to the GUIX engine to render an updated image using the gx_system_event_send API.

These common steps are illustrated in a typical operational flow in the following figure:

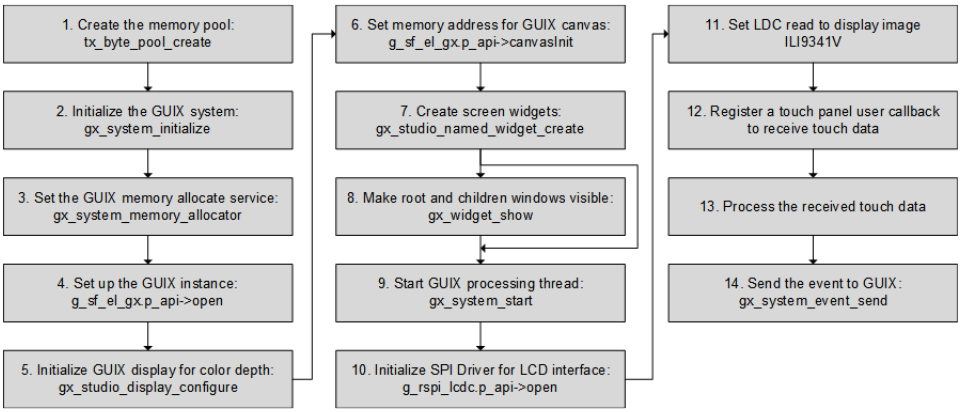


Figure 415: Flow Diagram of a Typical GUIX GX_SRC Framework Module Application

The GUIX Studio project includes a callback function for the Event Function of the main window in this project. When this callback, thermos_screen_event_handler, is invoked by GUIX, it computes how to adjust the needle on the thermostat in the image and forwards that information back to GUIX for how to render an updated image based on a user touch event:

1. Determine event type from the event_ptr input from GUIX
2. Get the current needle angle data from GUIX using the gx_circular_gauge_angle_get API.
3. Send the updated information about the event to GUIX using the gx_window_event_process API.

The GUIX Studio project includes a callback function for the Draw Function of the "plus" and "minus" buttons in the main screen. When this callback, custom_pixelmap_button_draw, is invoked by GUIX, it redraws the buttons to add the "+" or "-" depending on which button is pressed. This does not have a great deal of functionality per se, but demonstrates how the draw function callback works with GUIX.

4.3.6 LevelX Port Framework on sf_el_lx_nor

4.3.6.1 Port LevelX Framework Module Introduction

The Port LevelX Framework implements the driver APIs (sector read, sector write, block erase and block erased verify) mandated by Azure RTOS LevelX NOR component. In addition to implementing the LevelX NOR driver method, it is also responsible for updating the flash geometry to perform

LevelX NOR initialization. It uses the sf_memory_api implementation to perform operations on the NOR flash.

Unsupported Features

LevelX NAND is not supported in this version of SSP.

Port LevelX Framework Module Features

- Implements LevelX NOR driver APIs to perform operations on NOR flash memory device.
- Set up NOR flash geometry.

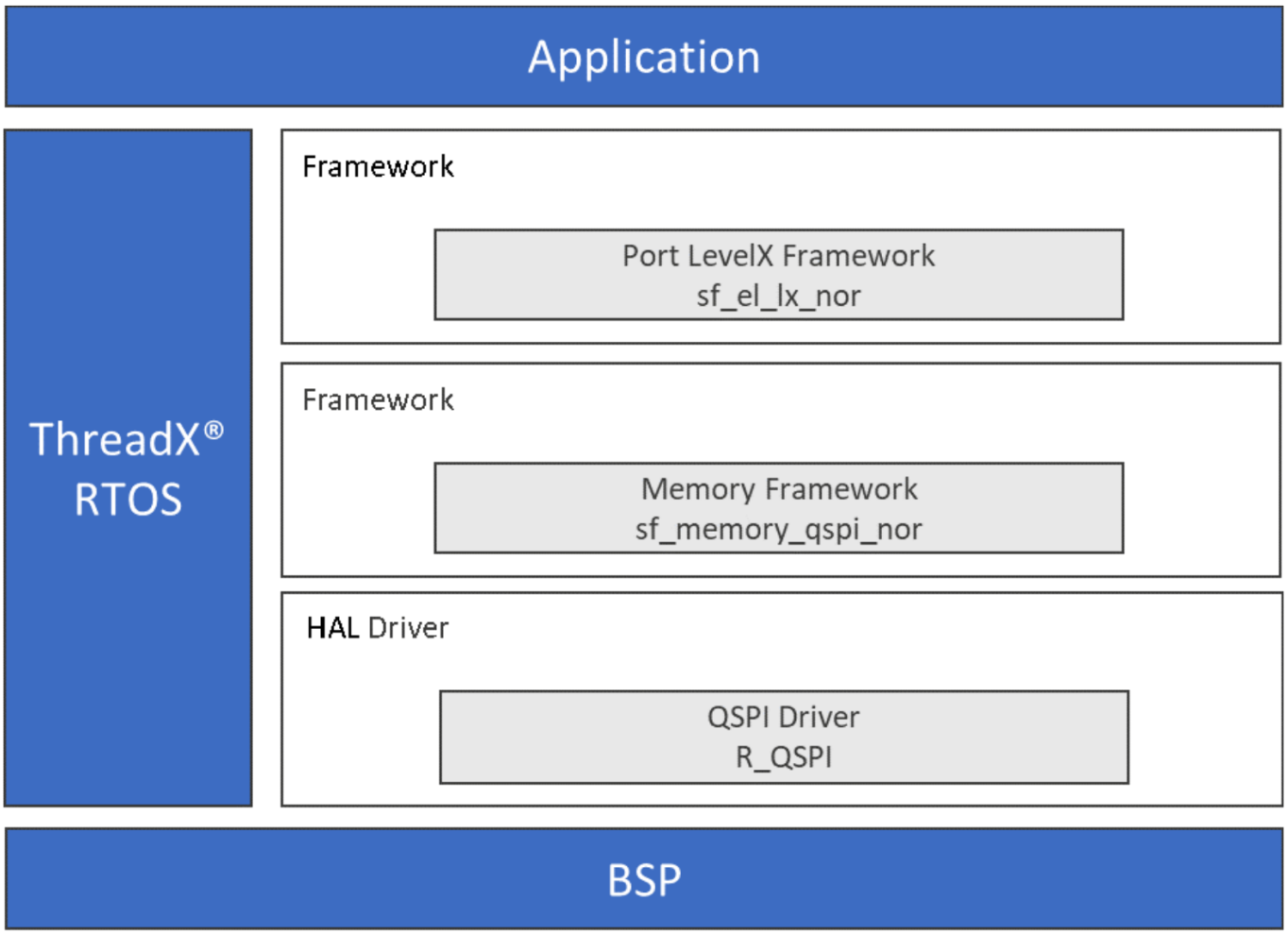


Figure 416: Port LevelX Framework Module Block Diagram

4.3.6.2 Port LevelX Framework Module APIs Overview

The Port LevelX Framework defines API functions to open, read from, write to and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Port LevelX Framework Module API Summary

Function Name	Example API Call and Description

SF_EL_LX_NOR_Open	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_Open(g_sf_el_lx_nor0.p_ctrl, g_sf_el_lx_nor0.p_cfg); Initializes lower-level driver initialization function.
SF_EL_LX_NOR_Read	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_Read(g_sf_el_lx_nor0.p_ctrl, p_flash, p_dest, word_count); This is responsible for reading a specific sector in a specific block of the NOR flash. All error checking and correcting logic is the responsibility of this service.
SF_EL_LX_NOR_Write	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_Write(g_sf_el_lx_nor0.p_ctrl, p_flash, p_src, word_count); This is responsible for writing a specific sector into a block of the NOR flash. All error checking is the responsibility of this service.
SF_EL_LX_NOR_BlockErase	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_BlockErase(g_sf_el_lx_nor0.p_ctrl, block, erase_count); This is responsible for erasing the specified block of the NOR flash.
SF_EL_LX_NOR_BlockErasedVerify	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_BlockErasedVerify(g_sf_el_lx_nor0.p_ctrl, block); This is responsible for verifying that the specified block of the NOR flash is erased.
SF_EL_LX_NOR_Close	g_sf_el_lx_nor0.p_api->SF_EL_LX_NOR_Close(g_sf_el_lx_nor0.p_ctrl); This is responsible for closing the driver properly.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_ASSERTION	p_ctrl or p_cfg in NULL.
SSP_ERR_ALREADY_OPEN	Driver is already in OPEN state.
SSP_ERR_INVALID_ARGUMENT	Requested range can't fit in the flash address range.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for writing.
SSP_ERR_NOT_ERASED	The block in not erased properly.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.6.3 Port LevelX Framework Module Operational Overview

The Port LevelX Framework is simply an abstract interface using function pointers instead of direct function calls. Functions are called between LevelX or FileX and the SSP memory interface implementations such as sf_memory_qspi_nor. Memory adaptation drivers, such as sf_memory_qspi_nor, are accessed through the Port LevelX Framework and provide device specific code needed to perform data I/O operations.

Port LevelX Framework Module Important Operational Notes and Limitations

Port LevelX Framework Module Operational Notes

None.

Port LevelX Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.6.4 Including the Port LevelX Framework Module in an Application

This section describes how to include the Port LevelX Framework Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Port LevelX Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Port LevelX Framework is g_sf_el_lx_nor0. This name can be changed in the associated Properties window.)

Port LevelX Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_lx_nor0 Port LevelX Framework on sf_el_lx_nor	Threads	**New Stack**> **Framework**> **LevelX**> **Port LevelX Framework on sf_el_lx_nor**

When the Port LevelX Framework on sf_el_lx_nor is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

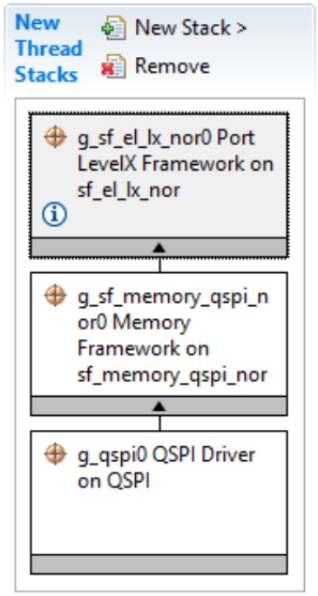


Figure 417: Port LevelX Framework Module Stack

4.3.6.5 Configuring the Port LevelX Framework Module

The Port LevelX Framework Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Port LevelX Framework Module on sf_el_lx_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_el_lx_nor0	Module name.
Event Callback	NULL	Name of the function to call when an event occurs.
Memory Partition Size Unit in (KB/MB)	Kilobyte (KB), Megabyte (MB) Default: Kilobyte (KB)	The units of memory.

Memory Partition Size	Positive Integer Number Default: 0	Size of the NOR Flash partition region.
Memory Partition Start Address(in hex format)	Positive Integer Number in Hex Default: 0x00000000	The starting address of the partition region.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Port LevelX Framework Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the Memory Framework on sf_memory_qspi_nor

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_memory_qspi_nor0	Module name.
Write of Erase Timeout (in ticks)	30000	Timeout ticks for waiting on write or erase to complete.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_qspi0	Module name.
Addressing Mode	3-BYTE, 4-BYTE Default: 3-BYTE	Addressing modes of flash memory For memory > 16 MB, 4-Byte addressing mode should be selected

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have

different default values and available configuration settings.

Port LevelX Framework Module Clock Configuration

The Port LevelX Framework module uses the QSPI peripheral which uses PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Port LevelX Framework Module Pin Configuration

To use the Port LevelX Framework module, the port pins for the QSPI peripheral must be set as needed. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection for the Memory Framework Module on sf_memory_qspi_nor

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select **Peripherals**> **Storage:QSPI QSPI0**

4.3.6.6 Using the Port LevelX Framework Module in an Application

The typical steps in using the Port LevelX Framework module in an application are:

1. Initialize the module using the lx_nor_flash_initialize API function (g_common_init calls it automatically).

2. Open the module for I/O operations using the LevelX NOR API function SF_EL_LX_NOR_Open (if using with FileX fx_media_open opens the media automatically).

3. Read the media as required using the SF_EL_LX_NOR_Read API function (lx_nor_flash_sector_read calls this automatically).

4. Write to media as required using the SF_EL_LX_NOR_Write API function (lx_nor_flash_sector_write calls this automatically).

5. A block can be erased using the SF_EL_LX_NOR_BlockErase API function.

6. A block can be verified whether it is erase or not using the SF_EL_LX_NOR_BlockErasedVerify API function.

Note

After a successful SF_EL_LX_NOR_Open call, all LevelX NOR APIs can be used, not only read and write.

These common steps are illustrated in a typical operational flow diagram in the following figure:

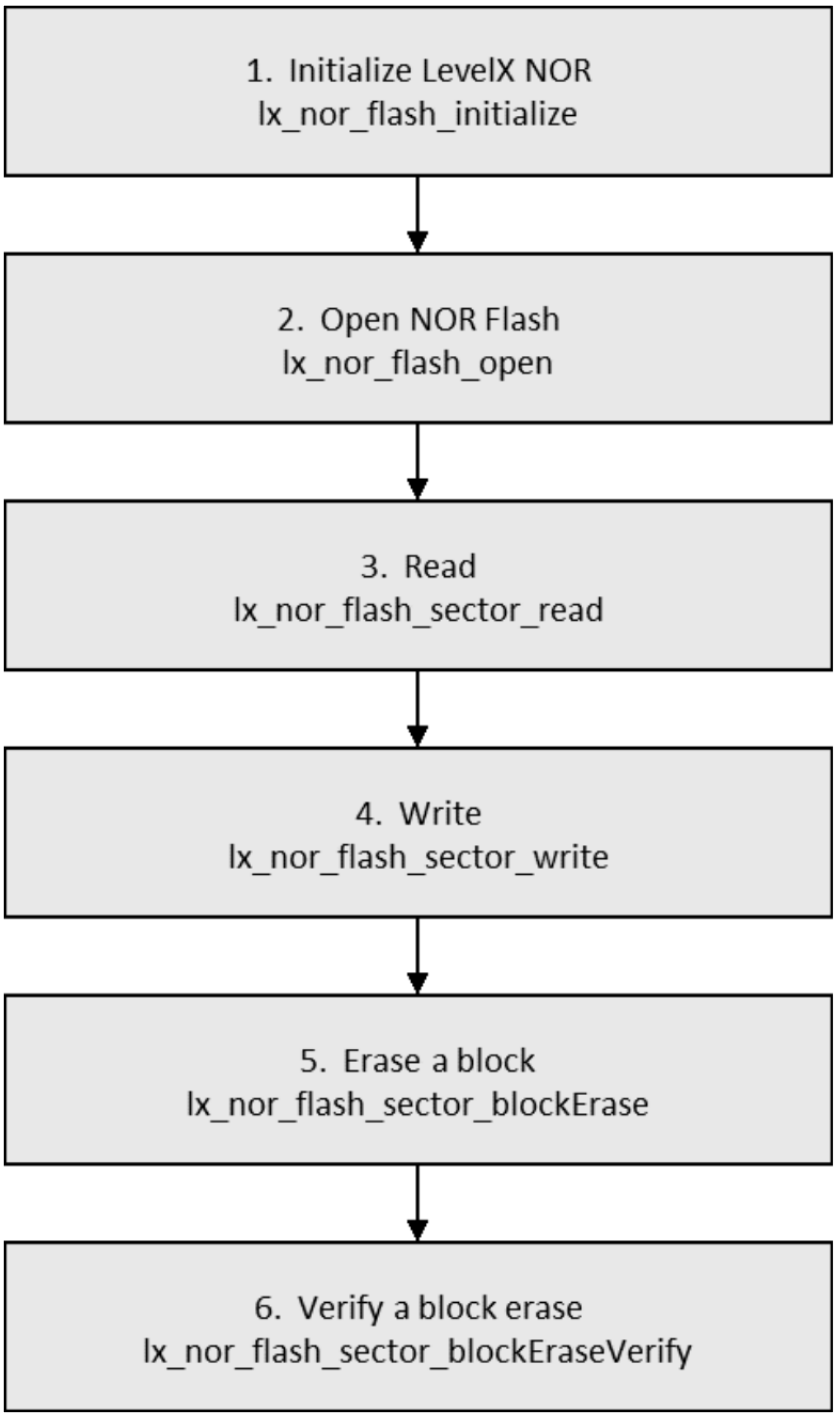


Figure 418: Flow Diagram of a Typical Port LevelX Framework Module Application

4.3.7 NetX Port Ether

4.3.7.1 NetX Port Ether Module Introduction

The Azure RTOS NetX Port Ether module (sf_el_nx) for NetX and NetX Duo is integrated into the SSP. Its function is to interface the NetX and NetX Duo software with the Synergy hardware. This module includes the MAC driver, the PHY driver, additional glue logic and utility functions.

Note

Unless otherwise stated, there is no difference in how this module works between NetX or NetX Duo projects.

NetX Port Ether Module Features

- High-level interface for NetX and NetX Duo for the Synergy Platform
- Channel Selection
- PHY Reset support
- Static MAC Address configuration
- Dynamic MAC Address configuration
- Callbacks are provided for unknown packet reception
- Selectable Ethernet Interrupt Priority
- Link status monitoring support
- Configurable Number of Receive/Transmit Buffer Descriptors
- Supports the use of an external PHY chip

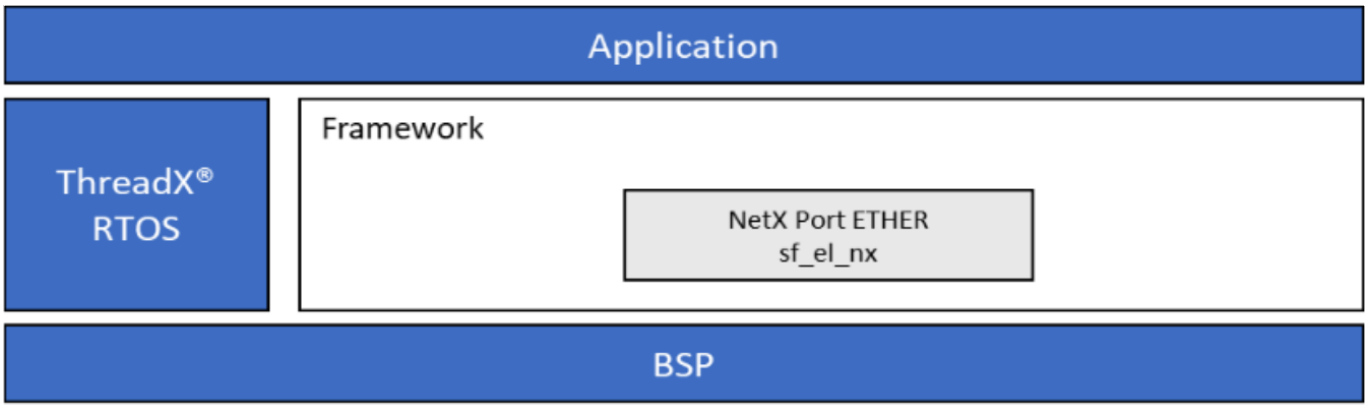


Figure 419: NetX Port Ether Module Block Diagram

4.3.7.2 NetX Port Ether Module APIs Overview

The NetX Port Ether module has a narrow API, used internally by NetX and by the module itself. It includes the Ethernet driver entry point (nx_ether_driver_eth0, nx_ether_driver_eth1), the Ethernet interrupt handler and other functions used internally by the module but externally visible.

4.3.7.3 NetX Port Ether Module Operational Overview

The NetX Port Ether module is a high-performance real-time implementation of the NetX Ethernet driver for the Renesas Synergy software and Synergy Ethernet IP.

Note

NetX assumes the existence of ThreadX and depends on its thread execution, suspension, periodic timers and mutual exclusion facilities.

Each IP instance in NetX has a primary interface which is identified by its device driver specified in the nx_ip_create service. The network driver is responsible for handling various NetX requests, including packet transmission, packet reception, and requests for status and control.

For a multi-home system, the IP instance can be configured for multiple interfaces, each with an associated network driver that performs these tasks for the respective interface. The network driver must also handle asynchronous events occurring on the media. Asynchronous events from the media include packet reception, packet transmission completion, and status changes. NetX provides the network driver with several access functions to handle various events. These functions are designed to be called from the interrupt service routine portion of the network driver. For IP networks, the network driver should forward all ARP packets received to the nx_arp_packet_deferred_receive internal function. All RARP packets should be forwarded to nx_rarp_packet_deferred_receive internal function. There are two options for IP packets:

If fast dispatch of IP packets is required, incoming IP packets are forwarded to nx_ip_packet_receive for immediate processing; this greatly improves NetX performance in handling IP packets. Otherwise, IP packets are forwarded to nx_ip_packet_deferred_receive. This service places the IP packet in the deferred processing queue where it is then handled by the internal IP thread, which results in the least amount of ISR processing time.

The network driver can also defer interrupt processing to run out of the context of the IP thread. In this mode, the ISR shall save the necessary information, call the internal function nx_ip_driver_deferred_processing, and acknowledge the interrupt controller. This service notifies IP thread to schedule a callback to the network driver to complete the process of the event that causes the interrupt.

Key Configuration Property Settings:

Multi-Channel

Multi-Channel determines which interface the ethernet driver is applied to. It may need to be modified from the default value of zero. See Section 5 "Configuring the NetX Port Ether".

Channel 0/1 Phy Reset Pin

PHY reset is supported through I/O port pins. This property depends on the value of the Channel property above.

Note

For different Synergy Kits different I/O PINs are being utilized, for this purpose it may need to be modified from the default value.

Static MAC Address Configuration

These properties set the device MAC address on the respective channel interface at compile time. To set the MAC address at run time, see the description of the Dynamic MAC Address Configuration callback property below.

Dynamic MAC Address Configuration

This property defines the user defined callback function which sets MAC address at runtime (network link initialization). The default value is NULL and the MAC address is assigned using the Channel MAC Address High/Low Bits settings.

Unknown Ethernet Packet Receive Callback

This property configures callback for letting user process unsupported/custom EtherType packets.

This call-back can be used in conjugation with nx_ether_custom_packet_send Ethernet API to send

and receive custom Ethernet packet types.

Name

This property names the instance of the NetX Port Ether driver. For an IP instance configured with multiple network interfaces, the application must add an instance of the driver and give it a unique name (or compile errors will result). The default name is g_sf_el_nx.

Ethernet Interrupt Priority

This property sets the driver interrupt priority. The default value is disabled. The dropdown list indicates valid and invalid priorities depending on the MCU target.

Link Status Monitoring

This property lets user choose link status monitoring method. There two options available: PHY Polling and PHY Interrupt (Uses LINKSTA Pin).

PHY polling utilizes an internal monitoring thread for auto negotiation to notify user when link is lost or established.

PHY Interrupt method requires PHY status pin to be connected to LINKSTA pin of Ethernet controller and utilize Ethernet interrupt to notify user when link is lost or established.

This notification though happens through the callback registered with IP instance.

Number of Receive/Transmit Buffer Descriptors

These properties set the number of buffer descriptors (BDs) for receiving and transmitting packets. When the Receive BDs are initialized, the driver allocates a packet for each BD. Therefore, the number of receive BDs should not deplete the IP instance packet pool from which it allocates packets.

The NetX Port Ether driver supports packet chaining for both receiving and transmitting packets (where packets are chained in the application layer). If a packet is received on the network that exceeds the size of the IP default packet pool payload, the driver can allocate additional packets and process the incoming packet as a packet chain.

Parameter Checking

This is low level error checking for each component of the project hardware layer. By default, it is set to Default (BSP) which means this property inherits the same property in the BSP.

NetX Port Ether Module Important Operational Notes and Limitations

NetX Port Ether Module Operational Notes

NetX Source Properties

There are two ways to modify NetX source properties- using the source code property directly in the source element or defining the source code symbol directly. For example, to change the number of physical network interfaces, one can either set the Maximum Physical Interfaces property in the NetX Source or NetX Duo Source element, or one can define the source code symbol NX_MAX_PHYSICAL_INTERFACES directly. In either case, it is still necessary to include the NetX and NetX Duo Source component, generate the project files and to rebuild the NetX library.

TCP Options Field Parameters

Maximum Segment Size

The Maximum Segment Size (MSS) is the maximum amount of bytes a TCP host can receive without being fragmented by the underlying IP layer. During TCP connection establishment phase, both ends exchanges its own TCP MSS value, so that the sender does not send a TCP data segment that is larger than the receiver's MSS. NetX TCP module will optionally validate its peer's advertised MSS value before establishing a connection. By default NetX does not enable such a check.

The nx_tcp_socket_mss_set() API sets a specified socket's Maximum Segment Size (MSS). The MSS value must be within the network interface IP Maximum Transfer Unit (MTU), allowing room for IP and TCP headers. This service should be used before a TCP socket starts the connection process. If the service is used after a TCP connection is established, the new value has no effect on the connection. To retrieve the MSS value use the nx_tcp_socket_mss_get() API after the TCP connection is established.

Using the External PHY with the NetX Port Ether Module

The NetX Port Ether Module can be used along with an external PHY chip, other than the on-board Micrel PHY chip (KSZ8081 and KSZ8091) found on Synergy Platform Kits. The following steps need to be performed to use a different external PHY chip:

1. Remove or undefine the macro BSP_BOARD_PHY_KSZ8081 or BSP_BOARD_PHY_KSZ8091 in the board BSP file (Example: For the S7G2_DK board, the BSP file is located in synergy/board/s7g2_dk/bsp_ethernet_phy.h).
2. Implement a new PHY driver for the specific PHY chip..Implement the functions defined in the interface file synergy/ssp/inc/framework/instances/sf_el_nx.h
3. Build and run the application

Note

The above steps also apply to existing projects using an external PHY.

Additional Information

Refer to the NetX User Guide for the Renesas Synergy™ Platform and NetX Duo User Guide for the Renesas Synergy™ Platform for additionaloperational details.

NetX Port Ether Module Limitations

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.3.7.4 Including the NetX Port Ether Module in an Application

This section describes how to include the NetX Port Ether Module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

The NetX Port Ether Framework is used as a lower level module and is not available to add as a separate stack. An example is illustrated in the following figure where a module must be selected to Add NetX Network Driver. The following figure shows the selection of the NetX Port Ether Framework

to complete the stack for the module.

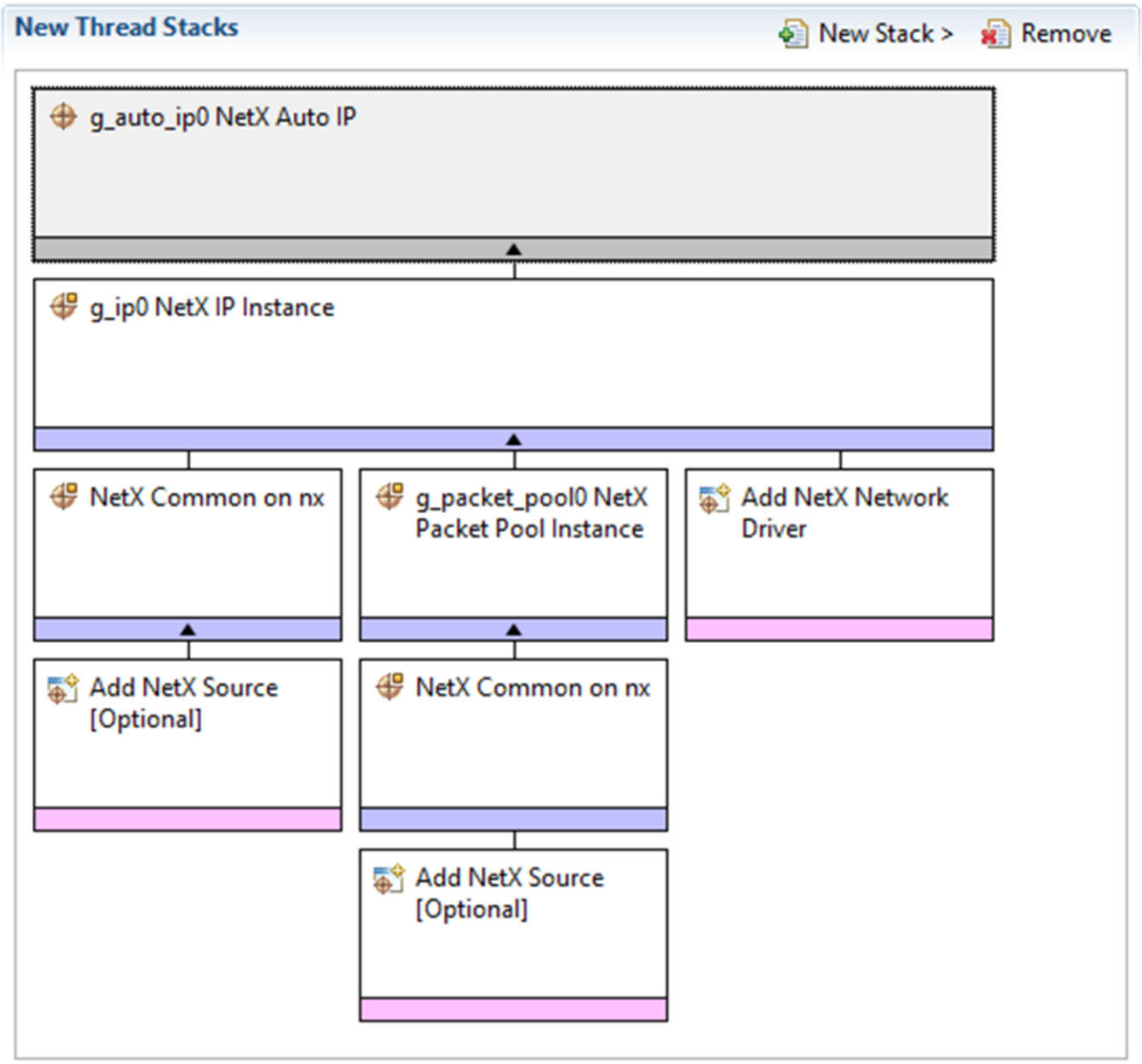


Figure 420: NetX Port Ether Module Stack

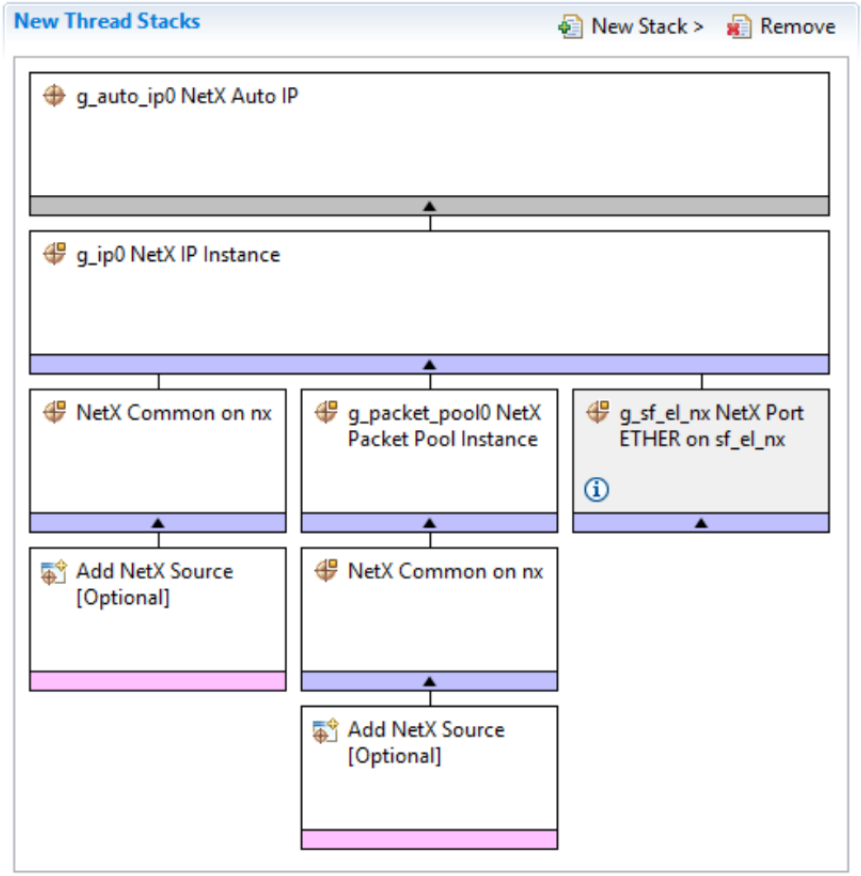


Figure 421: NetX Port Ether Module Stack

4.3.7.5 Configuring the NetX Port Ether Module

The NetX Port Ether Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Port Ether Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.

Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection.
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection.
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection.
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection.
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection.
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection.
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection.
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection.
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 lowest - not valid if using ThreadX) Default: Priority 12	Ethernet interrupt priority selection.
Link status monitoring method	PHY Interrupt (Uses LINKSTA Pin), PHY Polling Default: PHY Polling	PHY interrupt requires LINKSTA Pin connection to PHY.
Name	g_sf_el_nx	Module name.
Channel	0	Channel selection.
MAC address change callback	NULL	MAC address change callback selection.
Unknown packet receive callback	NULL	Unknown packet receive callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

4.3.7.6 Using the NetX Port Ether Module in an Application

The NetX Port Ether Module should be used in combination with NetX. Examples illustrating the modules use in an implementation for Auto-IP protocol use follow. Other protocols will follow a similar use flow. The steps performed by the Synergy Software Package (SSP) automatically in a NetX application support are:

1. Initialize the system with nx_system_initialize.
2. Create a packet pool with nx_packet_pool_create. This creates the IP default packet pool, to be used by the IP instance and the ethernet driver.

3. Create an IP Instance with nx_ip_create.
4. Enable ARP with nx_arp_enable.
5. Enable Link status change notify call back with nx_ip_link_status_change_notify_set.
6. Set gateway IP address for created IP instance nx_ip_gateway_address_set.
7. Create an AutoIP instance with nx_auto_ip_create.

The following steps are performed directly by the application to set up and run the Auto IP thread task.

1. Check if the network link is enabled with the nx_ip_interface_status_check API.
2. Verify the device does not have an IP address by calling the nx_ip_interface_address_get API.
3. Set the IP addreess notification callback by calling the nx_ip_address_change_notify API.
4. Start the Auto IP instance with the nx_auto_ip_start API.
5. Wait for the IP address change callback to set the flag that the IP instance has an IP address.
6. Stop the AutoIP task by calling the nx_auto_ip_stop API.
7. Verify the IP instance has a non-zero IP address by calling the nx_ip_interface_address_get API again.

These common steps are illustrated in a typical operational flow diagram in the following figure:

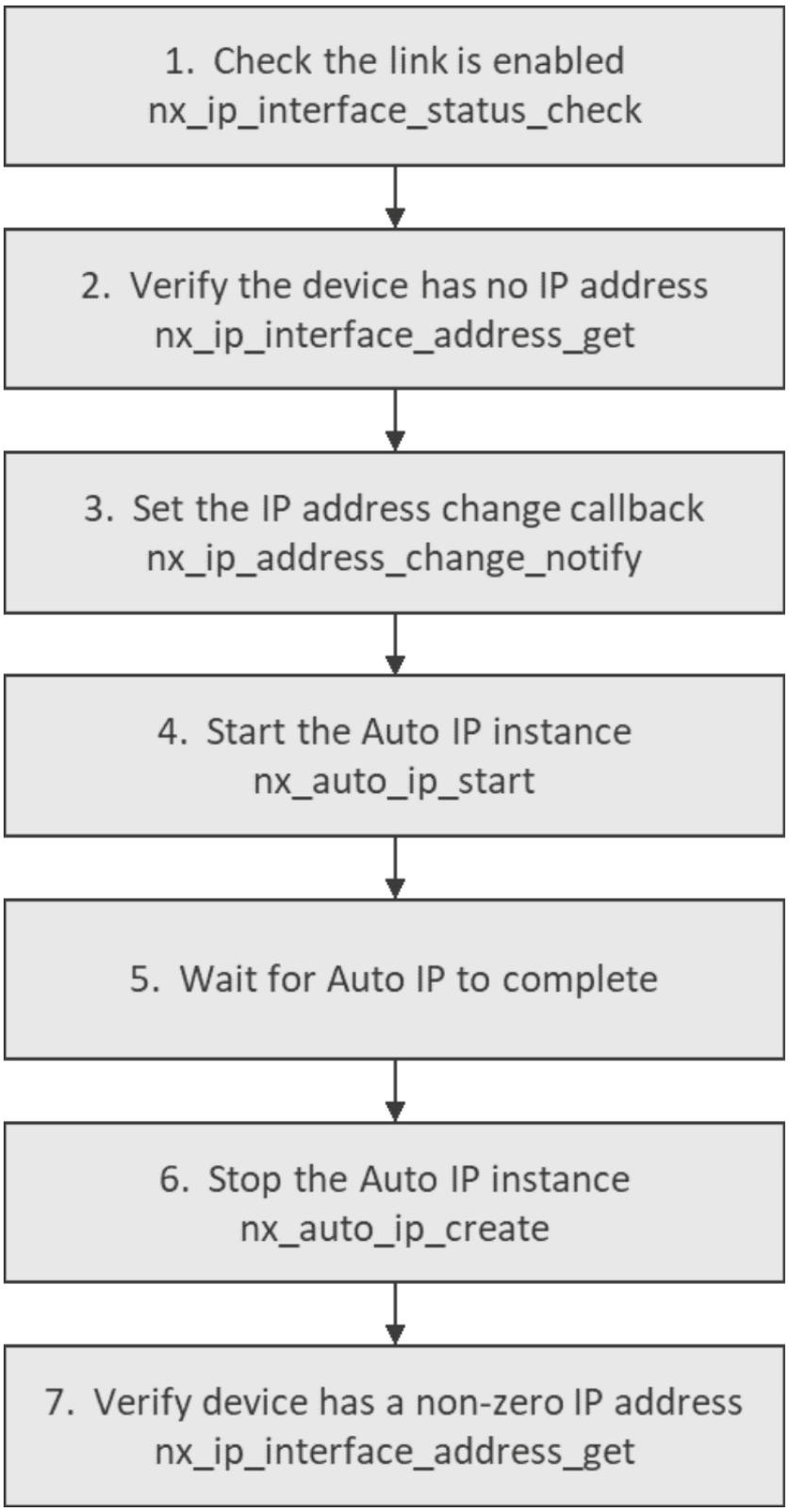


Figure 422: Flow Diagram of a Typical NetX Port Ether Module Application

4.3.8 NetX Port Using PPP

4.3.8.1 NetX Port Using PPP Module Introduction

The Azure RTOS NetX Port PPP module (nx_ppp) for NetX and NetX Duo is integrated into the SSP. Its function is to interface the NetX and NetX Duo software with the Synergy hardware. Point-to-Point Protocol (PPP) is a data link layer (layer 2) communication between two directly connected (point to point) devices. It can provide data transmission, authentication, and encryption functionality.

NetX Port Using PPP Module Features

- High-level interface for NetX and NetX Duo for the Renesas Synergy™ Platform
- CHAP authentication support
- PAP authentication support
- IPCP response support
- Handler provided for invalid packets

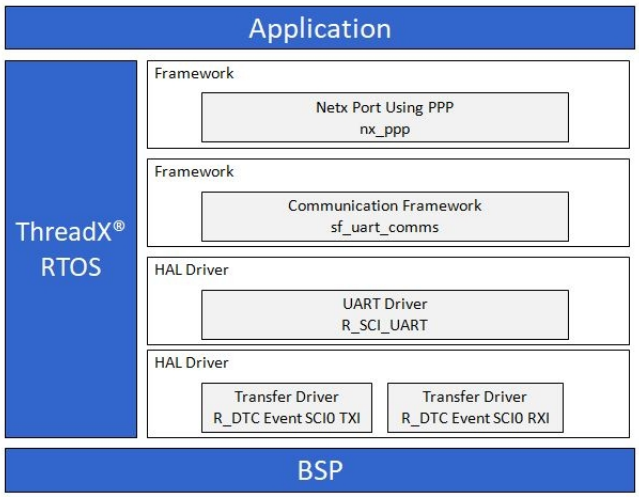


Figure 423: NetX Port Using PPP Module Block Diagram

4.3.8.2 NetX Port Using PPP Module APIs Overview

The NetX Port using PPP Module defines APIs for creating, authentication and assigns addresses. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo Port using PPP Module API Summary

Function Name	Example API Call and Description
nx_ppp_create	nx_ppp_create(&ppp_ptr, name, &ip_ptr, stack_memory_ptr, stack_size, thread_priority, &pool_ptr, ppp_non_ppp_packet_handler, ppp_byte_send); Creates PPP instance for given IP instance.

nx_ppp_dns_address_set	nx_ppp_dns_address_set(&ppp_ptr,& dns_address); Sets the primary DNS address for the PPP device.
nx_ppp_secondary_dns_address_set	nx_ppp_secondary_dns_address_set(&ppp_ptr, &secondary_dns_address); Sets secondary DNS address for the PPP device.
nx_ppp_ip_address_assign	nx_ppp_ip_address_assign(&ppp_ptr, local_ip_address, peer_ip_address); Assigns Local and Peer IP address.
nx_ppp_link_down_notify	nx_ppp_link_down_notify(&ppp_ instance, link_down_callback); Registers a callback for link down event.
nx_ppp_link_up_notify	nx_ppp_link_up_notify(&ppp_instance, link_up_callback); Registers a callback for link up event.
nx_ppp_nak_authentication_notify	nx_ppp_nak_authentication_notify(&ppp_ptr, nak_auth_callback); Registers a callback for authentication NAK received from peer.
nx_ppp_chap_enable	nx_ppp_chap_enable(&ppp_instance, get_challenge_values, get_responder_values, get_verification_values); Enables CHAP authentication for given PPP instance.
nx_ppp_pap_enable	nx_ppp_pap_enable(&ppp_instance, generate_login, verify_login); Enables PAP authentication for given PPP instance.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Valid PPP pointer.
NX_PTR_ERROR	Invalid pointer input.
NX_CALLER_ERROR	Invalid caller of this service.
NX_NOT_IMPLEMENTED	PAP logic was disabled via NX_PPP_DISABLE_PAP. CHAP logic was disabled via NX_PPP_DISABLE_CHAP.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.8.3 NetX Port Using PPP Module Operational Overview

The NetX PPP package requires the application to provide a serial communication driver. The driver must support 8-bit characters and may also employ software flow control. It is the application's responsibility to initialize the driver, which should be done prior to creating the PPP instance. In order to send PPP packets, a serial driver output byte routine must be provided to PPP (specified in the nx_ppp_create function). This serial driver byte output routine will be called repetitively in order to transmit the entire PPP packet. It is the serial driver's responsibility to buffer the output. On the receive side, the application's serial driver must call the PPP nx_ppp_byte_receive function whenever a new byte arrives. This is typically done from within the context of an Interrupt Service Routine(ISR). The nx_ppp_byte_receive function places the incoming byte into a circular buffer and alerts the PPP receive thread of its presence.

NetX Port Using PPP Module Important Operational Notes and Limitations

NetX Port Using PPP Module Operational Notes

If a modem is required for connection to the internet, some special considerations are required in order to use PPP. Basically, using a modem introduces additional initialization logic and logic for loss of communication. In addition, most of the additional modem logic is done outside the context of NetX PPP. The basic flow of using the NetX PPP with a modem is:

1. Initialize the Modem
2. Dial Internet Service Provider (ISP)
3. Wait for Connection
4. Wait for UserID Prompt
5. Start NetX PPP [PPP in operation]
6. Loss of Communication
7. Stop NetX PPP (or restart via nx_ppp_restart)

Additional Information

Refer to the NetX User Guide for the Renesas Synergy™ Platform and NetX Duo User Guide for the Renesas Synergy™ Platform for additional operational details.

NetX Port Using PPP Module Limitations

- No PPP Client XML support as it may not be useful.
- Current XML supports only PPP and not PPPoE.
- sf_comms_telnet, sf_ux_comms_v2 are not supported.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.3.8.4 Including the NetX Port Using PPP Module in an Application

This section describes how to include either or both NetX/NetX Duo Port using PPP module in an application using the SSP configurator.

Note

This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of

the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo Port using PPP module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

The NetX Port using PPP Framework is used as a lower level module and is not available to add as a separate stack. An example is illustrated in the following figure where a module must be selected to Add NetX Network Driver. The following figure shows the selection of the NetX Port using PPP Framework to complete the stack for the module.

NetX/NetX Duo Port Using PPP Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_nx_ppp0 NetX Port on nx_ppp	Threads	**New Stack> X-Ware> NetX> NetX IP Instance**
g_nxd_ppp0 NetX Duo Port on nxd_ppp	Threads	**New Stack> X-Ware> NetX Duo> NetX Duo Instance**

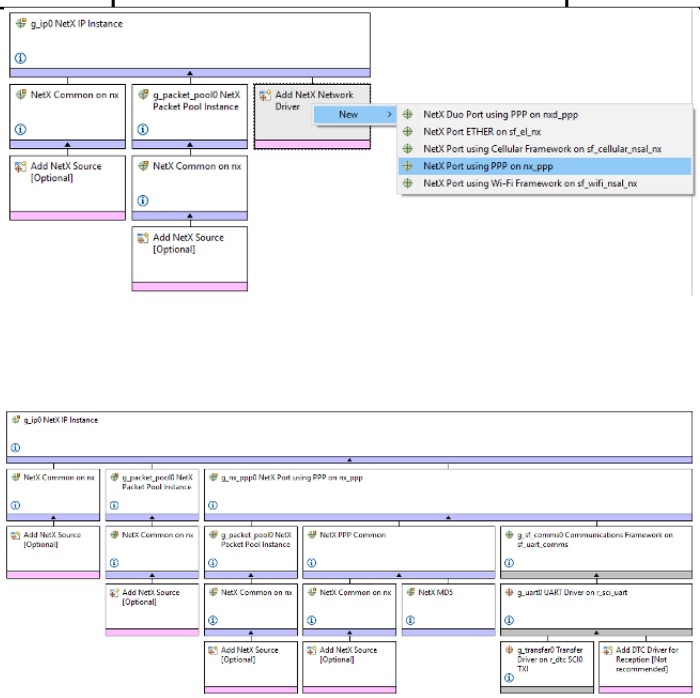


Figure 424: NetX Port Using PPP Module Stack

4.3.8.5 Configuring the NetX Port Using PPP Module

The NetX Port using PPP Module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Port using PPP Module

ISDE Property	Value	Description
CHAP authentication support	Enable, Disable Default: Enable	CHAP authentication selection.
PAP authentication support	Enable, Disable Default: Enable	PAP authentication selection.
DNS Option in IPCP response support	Enable, Disable Default: Enable	IPCP response selection.
Maximum DNS address request retires from the peer	2	Specifies the maximum number of DNS address request retries from the peer.
Data packet allocation timeout (seconds)	4	Timeout for data packet allocation.
Retry interval (seconds)	1	Specifies interval in seconds to wait before processing.
Maximum retries for reallocating packet	4	Specifies the maximum number of retries for reallocating packet.
Maximum retries for protocol request response	4	Specifies the maximum number of retries for protocol request response.
Maximum retries for LCP configure request	20	Specifies the maximum number of retries for LCP configure request.
Maximum retries for PAP authentication request	20	Specifies the maximum number of retries for PAP configure request.
Maximum retries for CHAP challenge message	20	Specifies the maximum number of retries for CHAP challenge message.
Maximum retries for IPCP configure request	20	Specifies the maximum number of retries for IPCP configure request.
Name	g_nx_ppp0	Name of PPP instance.
Internal thread priority	2	Internal thread priority selection.
Internal thread stack size (bytes)	2048	Thread Stack Size(bytes) selection.

Read thread priority	10	Thread Priority selection.
Peer IPV4 Address	192.168.0.111	IPv4 Address selection.
Name of invalid packet handler function	invalid_packet_handler	Name of invalid packet handler function selection.
Name of generated initialization function	nx_ppp_init0	Name of generated initialization function selection.
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Port Using PPP Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection

Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Link status change callback	NULL	Link status change callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes		Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the UART Communications Framework Module on sf_uart_comms

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if parameter checking is included.
Read Input Queue Size (4-Byte Words)	15	Buffer size for data reception queue. sf_uart_comms utilizes the ThreadX Queue for the queue management.
Name	g_sf_comms0	Name of UART communications framework module.
Name of generated initialization function	sf_comms_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the UART HAL Module on r_sci_uart

ISDE Property	Value	Description

External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to **Enable** and specify the configuration **Name of UART callback function for the RTS external pin control**.
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to **Disable** reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting **Disable** to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set **Disable** to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select **CTS** for this configuration parameter and enable the configuration **External RTS Operation** specifying the configuration **Name of UART callback function for the RTS external pin control**.
Name of UART callback function to be defined by user	NULL	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Defaualt: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Fallling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception; usually set **Falling Edge** to this configuration.
Noise Cancel	Enable, Disable Defaualt: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy™ hardware manual.
Bit Rate Modulation Enable	Enable, Disable Defaualt: Enable	Bit rate modulation enable selection.
Receive FIFO Trigger Level	One, Max Default: Max	Receive FIFO trigger level selection.

Receive Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selecition.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	Error interrupt priority selection.
Baud rate Percent Error	Value must be greater than 0.0 and less than 15.0 Default; 2.0	Maximum baudrate percent error allowed in order for the module to function.
UART Communication Mode	RS232, RS485 Default: RS232	UART communication mode selection, usually it is RS232 mode.
.UART RS485 Communication Mode	Half Duplex, Full Duplex Default: Half duplex	UART RS485 communication mode selection as half duplex or full duplex.
RS485 DE Port	00 to 11 Default: 09	Select the port number of Driver Enable pin.
RS485 DE Pin	00 to 15 Default: 14	Select the pin number of Driver Enable pin.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.

Software Start	Enabled, Disabled Defaualt: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_transfer1	Module name.

Mode	Normal	Mode selection.
Transfer Size	1 Byte	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX) Default: Disabled	ELC Software Event interrupt priority selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

4.3.8.6 Using the NetX Port Using PPP Module in an Application

The steps in using the NetX/NetX Duo Port using PPP module in a typical application are:

Auto Generated code to initialize NetX and NetX Duo Port using PPP in the Application (common_data.c)

- Create Packet pool using the nx_packet_pool_create API
- Create PPP using nx_ppp_create API
- Define IP address using the nx_ppp_ip_address_assign API
- Create IP instance using nx_ip_create() API
- Enable ARP using the nx_arp_enable API
- Enable TCP using the nx_tcp_enable API
- Enable UDP using the nx_udp_enable API
- Enable ICMP using the nx_icmp_enable API
- Enable IGMP using the nx_igmp_enable API

User Application Code (<thread>_entry.c)

- Call Link-up callback nx_ppp_link_up_notify() function (optional)
- Call Link-down callback nx_ppp_link_down_notify() (optional)
- Enable PAP using nx_ppp_pap_enable() API (optional)
- Wait for IP status check using nx_ip_status_check().
- Check IP and new mask assigned using nx_ip_interface_address_get().
- Wait till link is established using nx_ppp_status_get().
- Ping the client IP address to check the PPP connection using nx_icmp_ping().

4.3.9 NetX/NetX Duo Source

4.3.9.1 NetX and NetX Duo Source Module Introduction

TheNetX™ and NetX Duo™ source modules allow the developer to modify some of the key properties that control NetX operations. Adding the NetX or NetX Duo source component enables you in the Synergy configurator environment to customize the NetX and NetX Duo libraries, change values from default settings, and enable or disable certain features. Otherwise they must use the prebuilt NetX or NetX Duo library. In most projects beyond simple socket programs, you will typically want to customize their NetX or NetX Duo environment. Note that the ThreadX® source component is automatically added when adding a NetX or NetX Duo source component.

Without adding the NetX or NetX Duo source component, the Synergy ISDE configurator will use a prebuilt library with NetX and NetX Duo default settings.

4.3.9.2 NetX and NetX Duo Source Module APIs Overview

There are no APIs associated with the NetX or NetX Duo source module. This module is used to configure various NetX or NetX Duo properties.INote: Lower level drivers may return Common Error Codes. Refer to the *SSP User's Manual,* API References for the associated module for a definition of all relevant status return values.

4.3.9.3 NetX and NetX Duo Source Module Operational Overview

Using the NetX or NetX Duo source module is a bit different than using other SSP modules. The NetX or NetX Duo source module is used to configure networking operations; it doesn't provide API functions, callbacks or other typical module functions. There is no typical operational overview of the NetX or NetX Duo module. Refer to the NetX User's Manual, available from the Synergy Gallery for the operational details of NetX.

Using TraceX with NetX and NetX Duo

If TraceX is enabled in the ThreadX source component, it is automatically included when adding NetX or NetX Duo source components. The project containing the ThreadX and NetX, or NetX Duo library must be rebuilt. Otherwise the TraceX macros that log events will not be executed.

4.3.9.4 Including the NetX and NetX Duo Source Module in an Application

A network project generated in the e^2 studio configurator will automatically include an object Add

NetX Source.

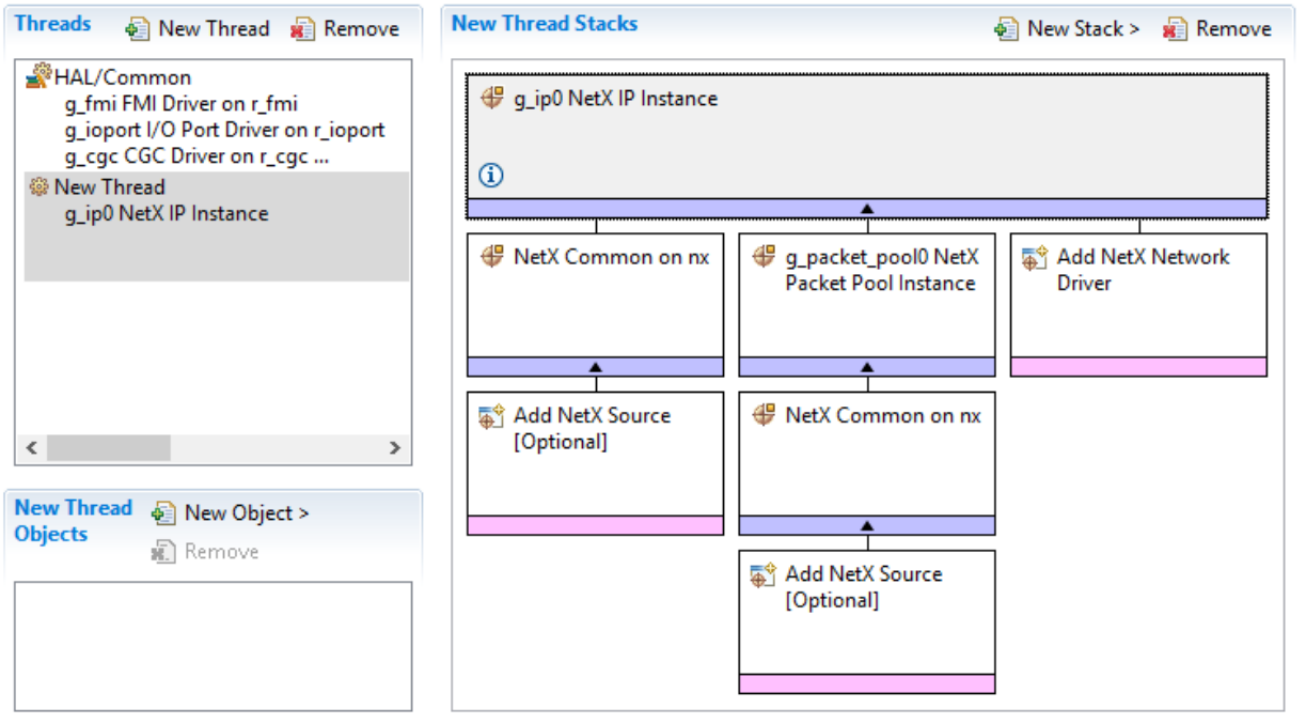


Figure 425: NetX and NetX Duo Source Module Stack

To add the NetX or NetX Duo source component to a project, click on the Add NetX Source (optional) or Add NetX Duo Source (optional) object) for NetX Duo in the e^2 studio configurator and choose New. If there are multiple Add NetX Source boxes, all of them will be updated automatically. Note that a ThreadX source component is added automatically.

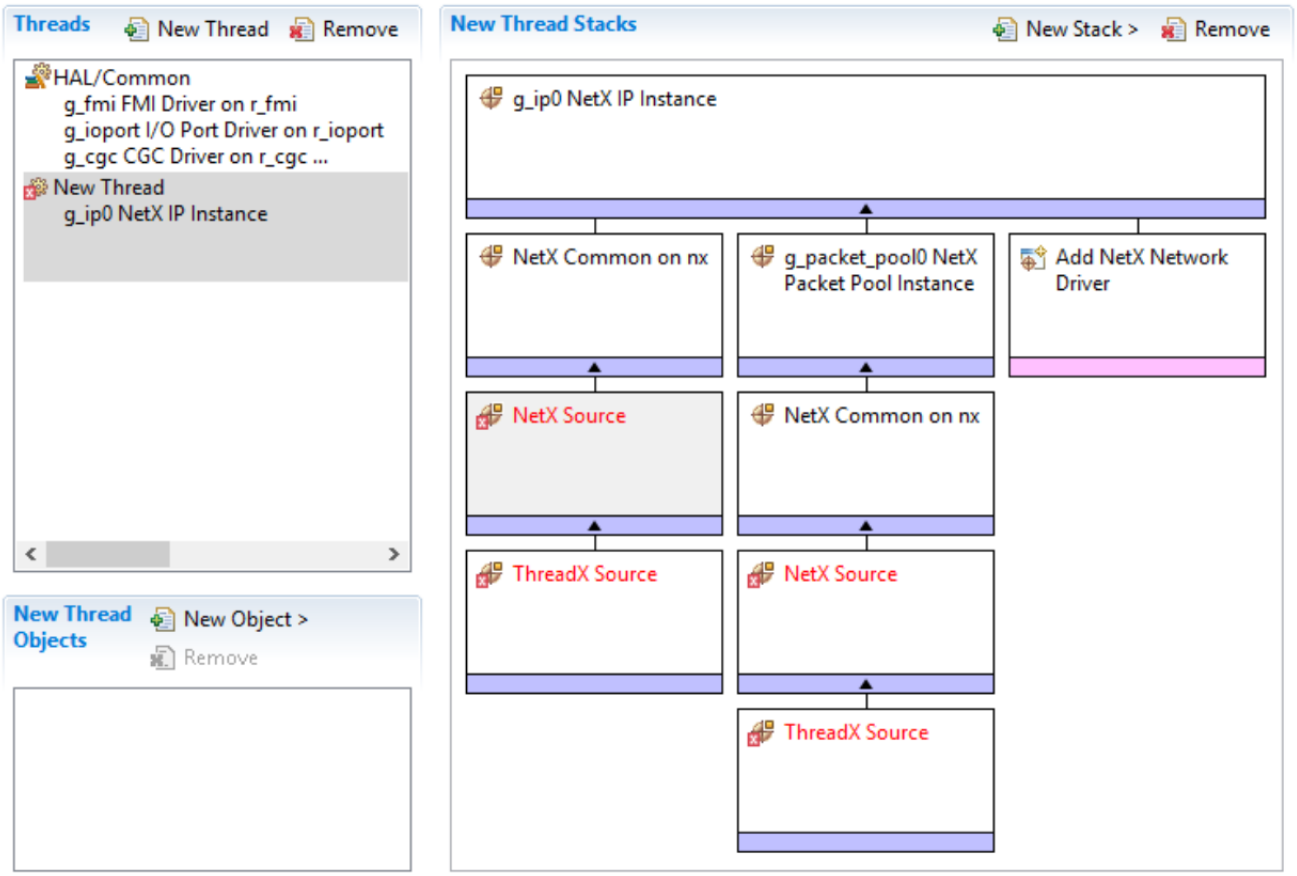


Figure 426: NetX and NetX Duo Source with ThreadX Source Stack

4.3.9.5 Configuring the NetX and NetX Duo Source Module

The list of properties below may not match the order of the NetX or NetX Duo property table. They are grouped according to general categories; ARP, TCP and IGMP for easier reference.

NetX or NetX Duo Source Module Configurable Properties

This section describes the configurable properties available to the NetX or NetX Duo source module and in what cases they can be changed from the default values to customize NetX or NetX Duo operations.

Note

*After changing NetX and NetX Duo property settings, the developer must click on the **Generate Project Content** button to update the project configurator in the ISDE. Then the NetX or NetX Duo library **must** rebuild the project. Changing a property (or applying a #define in the preprocessor list) without rebuilding the project will not affect any change. The Synergy™ Software Package (SSP) ISDE will use the previously built library.*

NetX or NetX Duo Properties Notes

These are some important notes that need to be understood by the developer before diving into the details of the large number of properties available for the NetX or Net Duo source module.

Default Settings. Most default settings are derived from RFC recommendations for the protocol relevant to the property.

IP Helper Thread Stack Size (bytes) – default value 1024. This is the stack size of the IP thread task which processes application NetX API calls, periodic events and deferred ISR events; packets received. There are some cases where a larger stack size will be required. Optimal stack size is usually determined empirically for an individual application.

IP fragmentation – default value enabled. To enable/disable fragmentation use the NetX/NetX Duo Source Fragmentation Option property. This property in the IP instance component is being deprecated.

__IP Helper Thread Priority – default value 3.__ This is the priority at which the IP thread task operates. Under some circumstances it might work best to increase this to 1 (highest valid priority). For example, if there are multiple network application thread tasks (DNS, HTTP, DHCP) in a project, then an IP helper thread at a higher priority (higher than network application threads) can service all network operations with higher priority, effectively improving responsiveness of the application thread tasks.

IP Layer Properties

Error Checking – default value enabled. It is generally enabled during development and debugging phase, and disabled when building a release version. When enabled, the NetX and NetX Duo include error checking services that will checks input and other parameters before calling the actual API. Some of the things it checks for are:

- NULL pointer input
- Invalid non-pointer parameters, such as an invalid IP address type or IP address equal to zero.
- Required configurable option must be enabled e.g. TCP must be enabled to use the nx_tcp_socket_receive API.
- The NetX or NetX Duo data structure IDs must match what is expected:

```
ip_ptr -> nx_ip_id == NX_IP_ID /* check the IP instance structure */
```

- Size of the data structure; the IP instance, matches the size of the data structure in the NetX library.

These last two checks guard against an application using a different version of the NetX or NetX Duo library than the application is using.

Static Routing – default value disabled allows certain destinations to be routed through a specific router. Normally a packet being transmitted is routed through the IP instance gateway/router as the next hop. When static routing is enabled, NetX checks the static routing table to determine if a packet's **next hop** address should go through a specific router rather than the IP instance gateway. The static routing table is limited to NX_IP_ROUTING_TABLE_SIZE entries, and there are various API such as nx_ip_static_route_add for managing the routing table. This is not commonly used but there are certain situations where it is necessary.

Physical Header and Physical Trailer – default value 16 and 4 bytes respectively: the type of network medium determines the physical (sometimes referred to as frame) header. An Ethernet network has a header of size 16 bytes. Wi-Fi has a larger header. The physical trailer is currently not being used for Ethernet but may need to be configured, depending on the physical network being used by the application. If the physical header is not set correctly, packets will not be assembled and most NetX packet send and receive services will not function correctly.

Maximum Listen Requests – default value 10 – This value is used by the TCP server applications to define the TCP listen queue size. When a TCP server socket binds a port (using the nx_tcp_server_socket_listen API), it requests a listen request on the IP instance. When it unlistens on that port, (using the nx_tcp_server_socket_unlisten API) the listen request is released, and the port is available to listen by another socket. This default value is not usually changed.

Driver Deferred Processing – default value Enabled – this feature enables the IP thread task to defer handling of the packet receive interrupts to the context of the IP thread task. Otherwise a packet receive event is handled in the context of the interrupt. This might lead to faster response to a specific packet but makes the overall system performance slower.

Loopback Interface – default value Enabled – if enabled, the NetX creates a loopback interface for sending and receiving packets to itself. Note that this is not counted as a physical interface. An application needs to be concerned about the number of network interfaces if they are using multiple physical interfaces. The loopback interface is not counted as a physical network interface (see below).

Maximum Physical Interfaces – default value 1 – The default is one network interface, generally referred to as the primary interface. The IP instance keeps a table of interfaces, and the primary interface is at index 0. For secondary interface(s), the Maximum Physical Interface value should be incremented by one for each secondary interface. Typical scenarios for multiple interface use is a router, with a local or private network on one interface and a global or public interface on the other. Another scenario is a device with two different network interfaces; ethernet and Wi-Fi.

To attach secondary interfaces to the IP instance, use the nx_ip_interface_attach API. When an interface is not referenced in an API, the action is on the primary interface. For example, nx_ip_status_check operates on the primary interface, while nx_ip_interface_status_checkperforms the operation on a specific interface.

Note: Some developers have used multiple IP instances to handle multiple physical networks. It is strongly not recommended to create multiple IP instances!

NAT (available in NetX Duo only) – default value disabled – NAT is a protocol that allows NetX Duo to map packets between the local/private and public/global networks, acting as a kind of router. To do this, the NetX Duo must be enabled to forward packets from a local host onto the global network with the host's global IP address and a port that the NetX Duo host determines. When NAT is enabled on the IP instance, NetX Duo can now forward packets in the manner. Note that NAT also requires that **Maximum Physical Interfaces** must be set to 2. SWIOT-5387 has an example project for setting up NAT.

Note

NAT is only available in NetX Duo.

Fragmentation option – default value enabled – Fragmentation occurs when the packet data exceeds the device MTU (maximum transfer unit) which is most typically 1518 bytes including frame header for Ethernet networks. An application should avoid fragmentation at the IP layer if possible, although there is not much one can do on the receiving side.

Another aspect of fragmentation to consider is the potential to deplete the packet pool used to receive packet fragments. When NetX is receiving many packet fragments, the packet pool used to receive these packets can easily be depleted, since packets cannot be returned to the packet pool until the entire packet is assembled by the IP layer and forwarded to the application. This can be remedied by increasing the number of packets in the packet pool at start up time (cannot be done at run time), if the board has sufficient memory to do so.

Note

This should not be confused with segmentation which is performed at the TCP layer. See TCP MSS Minimum for more details on segmentation.

Packet Header Pad Size – **default value 0 (no padding)** – Hardware operations on some boards require memory to be certain cache line is aligned. In those cases, the size of a NetX or NetX Duo packet instance must be aligned based on hardware requirements. For example, if the hardware requires the starting address of the data buffer to be 32-byte aligned and the size of the NX_PACKET header size is 56-bytes, defining the Packet Header ad size to be 8, can bring the starting address of the packet buffer to be 32-byte aligned.

Checksums – default value enabled – In NetX and NetX Duo, checksums for incoming (RX) and outgoing (TX) packets can be separately enabled or disabled. Checksums should not be disabled unless the RFC for that particular network protocol; UDP, TCP, or ICMP, indicate a zero checksum is allowable. Checksums generally need to be enabled. Disabling checksums might increase throughput but might also cause packets to be dropped by the other side of a connection.

Extended Notify Support – default value Disabled – When enabled, NetX will notify the application of various events related to TCP socket connections, above and beyond what NetX normally does (e.g. the callbacks specified in the nx_tcp_socket_receive_notify and nx_udp_socket_receive_notify APIs).

Extended Notify Support is required by NetX and NetX Duo BSD sockets.

The APIs enabled for this feature are listed below. They allow NetX to notify the application when NetX receives a TCP connection request, a TCP connection is completed, a TCP disconnect on a socket is completed, and to set the TCP socket state in the timed wait state. This is necessary for a TCP to operate in a non-blocking context.

nx_tcp_socket_syn_received_notify

nx_tcp_establish_notify

nx_tcp_disconnect_complete_notify

nx_tcp_timed_wait_callback

Source Address Check – default value Disabled – This checks all incoming packets for an invalid IP address. Specifically, it checks if 1) the IP address bits masked with the network mask does not equal the 1s complement of the network mask; 0xFF, 2) the IP address is not zero and the unmasked address bits do not equal zero, and 3) the address is not type D address (0xE0000000). This extra bit of processing at the IP level in NetX incurs a small performance penalty.

ICMP Address Check – default value Disabled – When enabled, the source address of ICMP packet is checked.

Maximum String Length – default value 1024 – This option specifies the maximum string length.

Enable random IP id – default value Disabled – When enabled, the default IP id is increased by one for each packet.

Enable VLAN – default value Disabled – When enabled, the VLAN feature is enabled.

IGPMP/MultiCast Properties

Maximum multicast groups – default value 7 – Defines the size of the multicast table in effect set the limit on the number of multicast groups that can be joined. Not generally modified.

IGMPV – default value Enabled – IGMPv2 is enabled by default. IGMPv2, unlike IGMPv1, allows group-specific join queries, leave messages and a method for determining which router will forward queries (instead of all IGMP routers on the network).

ARP Properties

ARP cache storage units – default in Bytes – Defines the ARP storage type in bytes. User can also select the option in Entries to define the number of ARP entries in table which will be aligned to the size of ARP.

ARP Cache size (in storage units) – default value 520 – Defines the size of the ARP cache in bytes. If user selects the storage units in Entries, then the value should be an integer. If user selects the storage units in Bytes, then the value should be a multiple of size of ARP (52 Bytes). This table holds all the ARP entries. If a table is full, no more ARP entries can be added till existing entries 'age' (see explanation of ARP Expiration Rate in this section) and are removed. There are some configurable options listed with the NetX/Duo Source element that can affect the number of ARP entries added to the table, such as ARP Auto ARP Entry and ARP Expiration Rate defined in this section.

The ARP cache size may need to be increased if the node is expected to communicate with large amount of the nodes on its local network.

ARP Auto ARP Entry – default value Enabled – This allows NetX to add an ARP entry when an ARP packet is received for which the table has no matching IP address. This will happen regardless of whether the response was directed to/from the NetX device. The idea behind auto ARP entry is to increase the efficiency of NetX by reducing the need to send ARP queries if the data is already in its ARP table. The downside is the ARP table can fill up and no more new ARP queries can be added. To prevent this, disable this feature. When disabled, NetX will only save ARP entries if the request was generated by NetX or directed to the NetX device.

This option is not required for most applications.

Alternatively, one can leave the Auto ARP Entry enabled, and set the expiration rate to non-zero to guarantee the entries will expire (and therefore be deleted) if not used. See **ARP Expiration Rate** for more details on the table entry expiration in this section.

ARP Expiration Rate – default value 0 seconds – By default ARP entries in NetX are **static**. They have no expiration rate and thus do not age. Otherwise, as an ARP entry ages, its timeout value decreases. When the timeout value reaches zero, the ARP entry is removed. If a new query pertaining to that ARP entry is received, the timeout value is reset. The timeout is also reset if the entry is accessed by the IP layer to transmit a packet. To determine if entries are being aged/deleted, an application can call the nx_arp_info_get API for statistics on aged entries as well as other useful statistics on ARP table management.

ARP Update Rate – default value 10 seconds – This is the interval between retransmission of ARP queries. When the number of retries reaches the ARP Maximum Retries, NetX abandons the attempt to find a physical mapping of IP address to a MAC address. There is no side effect to reducing this value to a smaller value.

ARP Maximum Queue Depth – default value 4 – This is the maximum number of IPv4 packets from the application trying to transmit packets for which NetX has no MAC address mapping. These packets are enqueued while NetX tries to find the MAC address mapping. When the queue is full, the

oldest packet is removed. To determine if this is happening, an application can check the nx_ip_transmit_resource_errors field in the IP instance data structure using the Expressions view in e^2 studio. Currently there is no API for obtaining this statistic.

This value does not need to be changed except under exceptional conditions. It is unusual for the queue to fill up using this default value. If the destination IP address is alive, an ARP response will be received by NetX and corresponding packets on the ARP queue will be transmitted. If the destination IP is dead, it does not make sense to queue any more packets to be sent to that address.

ARP Maximum Retries – default value 18 – This is the number of times NetX retransmits an ARP query for an IP address mapping before it gives up. There is no side effect to reducing this value to a smaller value.

ARP Defend by Reply – default value Disabled – This is intended for use with hosts who need to keep their current IP address.

Normally when a host gets an ARP packet with a matching source IP address but different MAC address, NetX may broadcast an ARP request to **announce** that it owns the IP address. It can only do so if it has not received a conflicting ARP packet within the interval of time specified by the **ARP Defend Interval** property (defined in this section). If it does send a defend reply, it resets the timeout to wait before responding to the next ARP conflict, if one occurs. If this is not the first conflicting ARP packet the host has seen, and the time recorded for the previous conflicting ARP packet is within the **ARP Defend Interval**, then the host MUST immediately cease using this address. This is necessary to ensure that two hosts do not get stuck in an endless loop with both hosts trying to defend the same address.

For more details on ARP conflicts, see *RFC 5227 IPv4 Address Conflicts Section 2.4 (b)*.

When **ARP Defend by Reply** is defined, an ARP reply is broadcast in addition to the one sent once the **ARP Defend Interval** expires. There is no requirement for a wait interval to expire for this ARP defend packet. This property is used because Windows XP ignores ARP request packets sent out when the **ARP Defend Interval** timeout expires.

ARP Defend Interval – default value 10 seconds – this is the interval during a NetX host that may not send an ARP defend packet if it receives a conflict ARP packet. 10 seconds is the default value recommended in the RFC 5227 for handling IPv4 address conflicts. After NetX sends the ARP defend packet, it resets the timeout back to this value.

ARP Mac Change Notification – default value Disabled – If NetX receives an ARP packet whose MAC address matches an entry in the ARP cache table, and this feature is enabled, NetX will call the ARP collision handler for the application to examine the packet and decide what to do with it. Without this feature, NetX will update the entry in its ARP table to the new MAC address. This behavior, that is a normal ARP protocol, can be taken advantage of in what is called the **man in the middle attack**, ARP cache poisoning or **spoofing**. This enables an attacker to redirect packets from one host to another by altering the MAC address information in the ARP table. NetX has internal handlers for these situations so applications need not handle them.

TCP Properties

Regarding the following group of rate setting properties, NetX timing is based on the NX_IP_PERIODIC_RATE setting. NX_IP_PERIODIC_RATE is derived directly from TX_TIMER_TICKS_PER_SECOND. The latter defaults to 100 but can be user defined, optionally in tx_port.h. If it is, NX_IP_PERIODIC_RATE is set to that value. If it is not, then NetX defaults it to 100 ticks (10 msec/tick).

TCP Fast Timer Rate – default value 10 – This determines the interval on how the fast timer periodic executes in NetX. If set to 10, and NX_IP_PERIODIC_RATE is set to 100, the fast periodic timer executes every 100 msec.

This timer is used to decrement the delayed ACK timeout (see **TCP ACK Timer Rate**) and the socket timeout (see **TCP Transmit Timer Rate**) for when to retransmit the ACK and data packets respectively. Increasing the **TCP Faster Timer Rate** shortens intervals between how the fast timer executes. In fact, it can potentially degrade performance for the extra overhead of processing the fast periodic timer more often.

TCP Retransmit Timer Rate – default value 1 (1 second) – This value is used to set the TCP socket timeout value. When a TCP socket sends or receives a SYN packet, or sends a data packet, it waits for an acknowledgment (ACK). If the socket timeout expires before receiving an ACK, the timeout is reset (and the packet retransmitted) up to a maximum of the **TCP Maximum Retries**. After that, NetX closes the connection.

It is generally not recommended to reduce this value. It will not make the TCP transactions happen quicker.

TCP ACK Timer Rate – default value 5 – This determines the interval between what NetX retransmits an ACK for missing data (or ACK probe). The NetX fast periodic timer decrements this timeout on each iteration of the faster timer periodic. If it expires, NetX sends another ACK packet. If a data packet is received, NetX resets the timeout. When data is sent from the NetX, it will also reset the timeout. If it expires, NetX increments the number of retries and resends an ACK. When the maximum number retries is reached (see **TCP Maximum Retries** defined in this section), NetX closes the connection.

A setting of 5, with NX_IP_PERIODIC_RATE set to 100, yields a delayed timeout of 200 msec. The greater the NX_TCP_ACK_TIMER_RATE the smaller the ACK timeout. Increasing the value does not increase performance or response time to the **NetX ACK**. It only sends them after a shorter interval.

TCP Maximum Retries – default value 10 – When a socket timeout expires without receiving a response from the TCP peer, the socket timeout is reset up until the number of retries equals the **TCP Maximum Retries**.

TCP Retry Shift – default value 0 – This is the bit shift applied to the retransmission interval. The default value of zero keeps the interval constant between socket retries (to get a response from the TCP peer). If it is set to 1, it doubles the interval; bit-shifts the timeout value by 1. This value is not often modified.

TCP Maximum TX Queue – default value 20 – This is the maximum number of packets that NetX will enqueue on a TCP socket for transmission and retransmission. A packet is enqueued when the socket is waiting to receive an ACK for the data, or it is waiting for the receive window of the other side to increase so it can send the data. For applications using smaller packet pools; limited memory resources, this value can be reduced to prevent packet pool depletion where a significant number of packets are sitting on the transmit queue, unavailable for other packet transmissions.

When the TCP socket has reached the maximum number of packets it can enqueue, the tcp socket will send call returns an NX_TCP_QUEUE_DEPTH error. If it cannot send a packet because the TCP receive window is too small it returns an NX_WINDOW_OVERFLOW error.

TCP Keepalive – default value disabled – The Keepalive feature starts a timer on the TCP socket in the established state; connected to a peer. When the timer expires, NetX sends a Keepalive ACK to the peer. Receiving a SYN or ACK packet, an ACK packet in response to NetX device's Keep Alive Ack, or any TCP packet with a valid sequence number resets the timer and the retry count.

When NetX initiates a Keepalive ACK exchange, it sends a Keepalive ACK packet with the ACK packet's sequence number decreased by one. This is how a TCP peer can distinguish a keepalive ACK from an ACK that indicates 1 byte of data has been sent.

When Keepalive is enabled on a socket, NetX also periodically checks if the other side has sent a Keepalive ACK.

If the Keepalive timer expires without a response from the peer, the number of Keepalive retries is incremented. If the retries reaches the TCP Keepalive Retries maximum, NetX deems the connection broken and terminates it. Without Keepalive, a TCP connection can persist indefinitely if neither side is transmitting a packet. In that situation, there is no way to know if the socket connection should be closed or remain open.

TCP Keepalive Retries – default value 10 - After 10 tries to get a response to a keepalive ACK, NetX resets **closes** the connection.

TCP Keepalive Initial – default value 7200 seconds (2 hours) – this is the interval before the first Keepalive is sent when a connection is completed, or a response is received from a previously sent Keepalive packet.

TCP Keepalive Retry – default value 75 seconds – NetX waits for the time specified by this option before resending another Keepalive packet, unless it has received a response to a previously sent Keepalive packet from the TCP peer.

TCP Window Scaling – default value Disabled – This feature allows a TCP receive window to exceed 65k up to a theoretical maximum of 1,073,725,440 bytes. When a NetX TCP Client socket initiates a connection, NetX computes a scaling factor based on the window size (set when the TCP socket is created). The window scaling values are exchanged during the TCP connection establishment phase. Note that the lack of window scaling option in peer's SYN packet is an indication that the peer does not support window scaling. In this situation, window scaling is not used for this connection.

TCP Maximum Out of Order Packets – default value disabled – If enabled, this option sets the maximum number of out of order packets that can be stored on the TCP socket receive queue. Subsequent out of order packets received are dropped if the socket receive queue has the maximum out of order packets. Eventually the NetX host should send an ACK indicating the missing data to the sender and get the dropped packet retransmitted. At this point, NetX can quickly process all the rest of the data on the receive queue and release the packets.

This feature is useful in the following scenario:

If a packet is lost or dropped, and the sender keeps sending packets, the TCP socket must enqueue all the packets back to the missing packet on its receive queue. It cannot release any because it is waiting for the missing data to be retransmitted. If the depth of the receive queue is not limited, this can starve the packet pool, rendering the NetX host effectively unresponsive. This can happen in a **bursty** data transmission, where many packets are sent by the TCP peer. A dropped packet may subsequently lead to many packets enqueued on the TCP receive queue.

TCP MSS Minimum – default value 128 – MSS, or maximum segment size is the largest amount of TCP data that will not require fragmentation. The **minimum** MSS is the lowest MSS that a NetX TCP socket will accept from a TCP peer. If the MSS parsed from the TCP header option data is below this value, the connection is dropped.

The intended usage of this is when an application wants to reject connections with small MSS values.

TCP ACK every N Packets – default is disabled – To enable this feature, enter a positive number. NetX sends an ACK out for every other data packet with new data, not retransmitted data. This is usually the optimal setting to minimize network traffic, packet processing and keep the TCP peer advertise window up to date. Note that if **TCP Immediate ACK** is enabled, this value is overridden; set to 1 automatically. Increasing the value increases the possibility the other side must wait on the **nth ACK** to know that the advertise window has increased enough to send more data. This may result in slower network throughput. The optimal setting is generally based on testing.

This ratio of ACKs per N packets is associated with the **delayed ACK** logic in TCP. For applications like Telnet, a higher N can greatly optimize the work on the Telnet Server receiving many 1 byte packets.

TCP Immediate ACK – default value disabled – If enabled, NetX sends an ACK for every packet of new data received. This is useful where a delayed ACK is not desirable.

Reset Disconnect – default value Enabled – When enabled, this allows an application to call a disconnect on a TCP socket with a zero wait option without sending a RST packet. It simply sends a FIN packet to indicate it is initiating a disconnection and closes the socket. It does not wait for an ACK or FIN ACK from the TCP peer. This is useful for hosts that do not want to wait to close a socket, such as servers wishing to free up sockets for the next Client request, and can do so without indicating something is wrong like a RST packet usually does.

RX Size Checking – default value Enabled – If enabled, NetX checks that a received packet has at least enough room for the IP or transport layer header; TCP, UDP, IGMP and ICMP depending on where the packet is being processed. In the IP layer, for example, NetX checks if the packet has at least enough room for the IP header. In the TCP layer, NetX checks if the packet has at least enough room for the TCP header.

As a packet travels up the stack in NetX, the packet **prepend** pointer and packet length are adjusted for each network layer. When the packet passes from the IP layer to the TCP layer, it moves the prepend pointer to the start of the TCP header, and subtracts the size of the packet length by the size of the IP layer. Similarly, going from the TCP layer to the application layer, the prepend pointer is moved past the TCP header to the application data or header in the case of something like an HTTP packet. The packet length is subtracted by the size of the TCP header.

This internal manipulation of the packet eliminates the need for the application adjust pointers and data size. An application or NetX protocol will typically make a socket call such as the nx_tcp_socket_receive API. If any packets are waiting on a socket receive queue, the application receives the packet and knows where the data is located and how much there is. The application can use the_nx_packet_length_getAPI (preferably) to obtain the packet data size, or access the nx_packet_length field in the NX_PACKET instance directly.

TCP/IP offload feature – default value disabled – If enabled, TCP/IP offload feature will be enabled. This feature enables NetX Duo to support a network interface card that offers TCP/IP service on the hardware. Certain WiFi modules offer TCP/IP processing on the module, and applications on MCU send and receive packets through APIs to access its TCP/IP stack. With this feature enabled, developers can run native NetX Duo applications directly.

Configure the "Enable the extra capability of the link driver" property to define **NX_ENABLE_INTERFACE_CAPABILITY** to enable the TCP/IP offload feature. This property is **disabled** by default.

Disable Assertion – default value disabled – If enabled, Assert is disabled.

Assert Fail – default value disabled – If enabled, the assert fail process is defined.

NetX and NetX Duo Statistics Properties

If enabled, NetX keeps statistics on its internal operations. These statistics are at the component level, like IP, TCP, ARP and Packet (pool). For TCP and UDP, there are statistics for all TCP/UDP transmissions and statistics per socket. Disabling these statistics reduces processing time slightly.

The value of these statistics is to be able to diagnose problems or optimize network performance without having to stop or interrupt program flow, or write tedious debug code.

Examples:

nx_packet_pool_info_get.c

If an application does not appear to be sending or transmitting packets, check the nx_packet_pool_empty_requests statistic. This is incremented every time nx_packet_allocatefails because no packets are in the packet pool. This is helpful if nx_packet_allocate is called from a void function or by a NetX protocol which may return a different value.

nx_ip_info_get.c

If an application is not receiving data, but packets are visible on a third-party packet trace, check the nx_ip_total_packets_received statistic to see if the data is forwarded at least as far as the IP level. Similarly, for not seeing packets from the NetX device on a packet trace is to check the nx_ip_total_packets_sent statistic.

The following is a partial list APIs for NetX statistics:

IP info – Statistics at the level of the IP layer receiving packets from the driver, and forwarding packets from the transport layer to the driver:

```
UINT  _nx_ip_info_get(NX_IP *ip_ptr, ULONG *ip_total_packets_sent,
ULONG *ip_total_bytes_sent, ULONG *ip_total_packets_received,
ULONG *ip_total_bytes_received, ULONG *ip_invalid_packets,
ULONG *ip_receive_packets_dropped, ULONG *ip_receive_checksum_errors,
ULONG *ip_send_packets_dropped, ULONG *ip_total_fragments_sent,
ULONG *ip_total_fragments_received)
```

nx_ip_interface_info_get.c is the same but specific to the specified interface.

ARP Info – Statistics on ARP packets sent and received, and ARP table statistics:

```
UINT  _nx_arp_info_get(NX_IP *ip_ptr, ULONG *arp_requests_sent,
ULONG *arp_requests_received,
                       ULONG *arp_responses_sent, ULONG *arp_responses_received,
                       ULONG *arp_dynamic_entries, ULONG *arp_static_entries,
                       ULONG *arp_aged_entries, ULONG *arp_invalid_messages)
```

Packet Pool Info – Statistics on available packets, empty packet (pool) requests, and invalid packet releases; invalid packet pool or packet pointer supplied.

```
INT  _nx_packet_pool_info_get(NX_PACKET_POOL *pool_ptr, ULONG *total_packets,
ULONG *free_packets,
                        ULONG *empty_pool_requests, ULONG *empty_pool_suspensions,
                        ULONG *invalid_packet_releases)
```

TCP Info – Statistics on the total number of TCP packets sent/received, number of connections and disconnections, dropped and retransmitted packets. The socket specific API includes the receive window size.

```
UINT  _nx_tcp_info_get(NX_IP *ip_ptr, ULONG *tcp_packets_sent, ULONG
*tcp_bytes_sent,
                        ULONG *tcp_packets_received, ULONG *tcp_bytes_received,
                      ULONG *tcp_invalid_packets,
ULONG *tcp_receive_packets_dropped,
                      ULONG *tcp_checksum_errors, ULONG *tcp_connections,
                     ULONG *tcp_disconnections, ULONG *tcp_connections_dropped,
                ULONG *tcp_retransmit_packets)
UINT  _nx_tcp_socket_info_get(NX_TCP_SOCKET *socket_ptr, ULONG *tcp_packets_sent,
ULONG *tcp_bytes_sent,
                ULONG *tcp_packets_received, ULONG *tcp_bytes_received,
                       ULONG *tcp_retransmit_packets, ULONG *tcp_packets_queued,
                 ULONG *tcp_checksum_errors, ULONG *tcp_socket_state,
                 ULONG *tcp_transmit_queue_depth,
ULONG *tcp_transmit_window,
                  ULONG *tcp_receive_window)
```

UDP Info – Statistics on the total number of UDP packets sent/received, dropped packets, improperly formed packets to send or receive. The socket specific API does not include the count of invalid packets received.

```
UINT  _nx_udp_info_get (NX_IP *ip_ptr, ULONG *udp_packets_sent, ULONG
*udp_bytes_sent,
```

```
                    ULONG *udp_packets_received, ULONG *udp_bytes_received,
                       ULONG *udp_invalid_packets,
ULONG *udp_receive_packets_dropped,
                     ULONG *udp_checksum_errors)
UINT  _nx_udp_socket_info_get (NX_UDP_SOCKET *socket_ptr, ULONG *udp_packets_sent,
ULONG *udp_bytes_sent, ULONG *udp_packets_received,
ULONG *udp_bytes_received, ULONG *udp_packets_queued,
                    ULONG *udp_receive_packets_dropped,
ULONG *udp_checksum_errors)
```

ICMP Info – Satistics on control message transmission, including count of unsupported ICMP messages, ping requests that timed out, and responses to ping request.

```
UINT  _nx_icmp_info_get(NX_IP *ip_ptr, ULONG *pings_sent, ULONG *ping_timeouts,
                    ULONG *ping_threads_suspended,
ULONG *ping_responses_received,
                      ULONG *icmp_checksum_errors,
ULONG *icmp_unhandled_messages)
```

IGMP Info – Statics on multicast groups joined, IGMP queries received, and IGMP reports sent.

```
UINT  _nx_igmp_info_get(NX_IP *ip_ptr, ULONG *igmp_reports_sent,
       ULONG *igmp_queries_received,
                ULONG *igmp_checksum_errors, ULONG *current_groups_joined)
```

4.3.10 Azure RTOS NetX Overview

4.3.10.1 Azure RTOS NetX Interface

Azure RTOS NetX is a high-performance real-time implementation of the TCP/IP standards designed exclusively for embedded ThreadX-based applications. The Azure RTOS NetX networking stack (nx) is integrated into the Renesas SSP. The Azure RTOS NetX Interface provides the means for SSP projects to access the capabilities of NetX.

For more information about NetX (including API references), refer to the *NetX User Guide* available

as part of the Azure RTOS Components documentation pack here: https://www.renesas.com/en-us/products/synergy/software/ssp.html

Unsupported Features

The following driver requests are not implemented in sf_el_nx:

- Get link speed
- Get duplex type
- Get error count
- Get receive packet count
- Get transmit packet count
- Get allocation errors
- User commands

Azure RTOS NetX with TraceX

If TraceX is enabled in the ThreadX source component, it is automatically included when adding the NetX source components. When enabled, the project containing the ThreadX and NetX library must be rebuilt or TraceX macros that log events will not be executed.

Azure RTOS NetX Protocol Modules

A wide variety of popular networking protocol functions are supported within the Azure RTOS NetX Interface for SSP. Each available protocol function has its own module overview section in this document. These overviews provide sufficient background to determine if the module provides the functions needed by the application.

Complete Application Projects (APs) are also available showing a working project for a typical NetX Protocol application targeted for a Renesas Synergy hardware kit. APs are available from the Renesas Synergy web site here: https://www.renesas.com/us/en/support/document-search?title=&doc_file_all_types%5BSample+Code%5D=Sample+Code&doc_category_tier_1=467666&doc_category_tier_2=469306&doc_category_tier_3=&doc_category_tier_4=&doc_part_numbers=&sort_order=DESC&sort_by=field_document_revision_date

Azure RTOS NetX Configuration Notes

When a protocol module is added to a project, a prebuilt library of the application code is added. For each protocol component, there is an analogous component ending in '_src' that contains protected source files. The '_src' component can be added in addition to the prebuilt library module. Do not add the '_src' component without the prebuilt library module.

If the NetX Source module is added to the project, the Properties window provides advanced configurations for the NetX source library. Highlight a configuration option to view a description of the option in the bottom left corner of the e^2 studio GUI. If the configuration option is empty, the default value is used. The default values of configuration options are defined in ssp/inc/framework/el/nx/nx_port.h. Refer to the Configuration Options chapter of the *NetX User Guide* for more information.

A detailed NetX and NetX Duo Source Module Overview is available in this document. It covers the configurable networking operations available within the source module. It is highly recommended that this is read and understood before attempting to modify any NetX or NetX Duo source configuration settings.

Azure RTOS NetX Limitations

- Azure RTOS NetX cannot be used in the same application as Azure RTOS NetX Duo. Only one or the other can be used per application.
- When using NetX BSD with the GCC compiler, build with the macro _POSIX_SOURCE defined to avoid compilation errors.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Azure RTOS NetX Supported Devices

- Refer to the associated SSP Release Notes for a complete list of the supported MCU devices.

4.3.11 Azure RTOS NetX Duo Overview

4.3.11.1 Azure RTOS NetX Duo Interface

Azure RTOS NetX Duo is a high-performance real-time implementation of the TCP/IP standards designed exclusively for embedded ThreadX-based applications. The Azure RTOS NetX Duo networking stack (nxd) is integrated into the Renesas SSP. The Azure RTOS NetX Duo Interface provides the means for SSP projects to access the capabilities of NetX Duo.

For more information about NetX Duo (including API references), refer to the *NetX Duo User Guide* available as part of the Azure RTOS Components documentation pack here: https://www.renesas.com/en-us/products/synergy/software/ssp.html.

Unsupported Features

The following driver requests are not implemented in sf_el_nx (SSP)

- Get link speed
- Get duplex type
- Get error count
- Get receive packet count
- Get transmit packet count
- Get allocation errors
- User commands

Azure RTOS NetX Duo with TraceX

If TraceX is enabled in the ThreadX source component, is automatically included when adding NetX Duo source components, the project containing the ThreadX and NetX Duo library must be rebuilt or TraceX macros that log events will not be executed.

4.3.11.2 Azure RTOS NetX Duo Protocol Modules

A wide variety of popular networking protocol functions are supported within the Azure RTOS NetX Duo Interface for SSP. Each available protocol function has its own module overview section in this document. These overviews provide sufficient background to determine if the module provides the functions needed by the application.

Complete Application Projects (APs) are also available showing a working project for a typical NetX Protocol application targeted for a Renesas Synergy hardware kit. APs are available from the Renesas Synergy web site here: https://www.renesas.com/us/en/support/document-search?title=&doc_file_all_types%5BSample+Code%5D=Sample+Code&doc_category_tier_1=467666&doc_category_tier_2=469306&doc_category_tier_3=&doc_category_tier_4=&doc_part_numbers=&sort_order=DESC&sort_by=field_document_revision_date

4.3.11.3 Azure RTOS NetX Duo Limitations

- Azure RTOS NetX Duo cannot be used in the same application as Azure RTOS NetX. Only one or the other can be used per application.
- When using NetX Duo BSD with the GCC compiler, build with the macro _POSIX_SOURCE defined to avoid compilation errors.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.11.4 Azure RTOS NetX Duo Supported Devices

- Refer to the associated SSP Release Notes for a complete list of the supported MCU devices.

4.3.12 NetX/NetX Duo Auto IP

4.3.12.1 NetX/NetX Duo Auto IP Introduction

The Auto IP Protocol is designed to dynamically configure IPv4 addresses on a local network without requiring a server, unlike the Dynamic Host Configuration Protocol (DHCP). The Auto IP uses address resolution protocol (ARP) for automatic IP address assignment and allocates addresses in the range of 169.254.1.0 through 169.254.254.255.

Note

Except for internal processing, the NetX Duo™ Auto IP module is identical in the application, set-up and running of an Auto IP session as the NetX™ Auto IP module.

NetX/NetX Duo Auto IP Module Features

- Compliant with RFC3927 and related RFCs
- Uses ARP probes to check for address conflicts
- Uses the collision handler notification in NetX to detect an address already in use
- Registers a valid Auto IP address with the IP instance
- Provides high-level APIs for:
    - Creating and deleting an Auto IP instance
    - Starting and stopping the Auto IP thread task
    - Specifying the network interface on which to run Auto IP

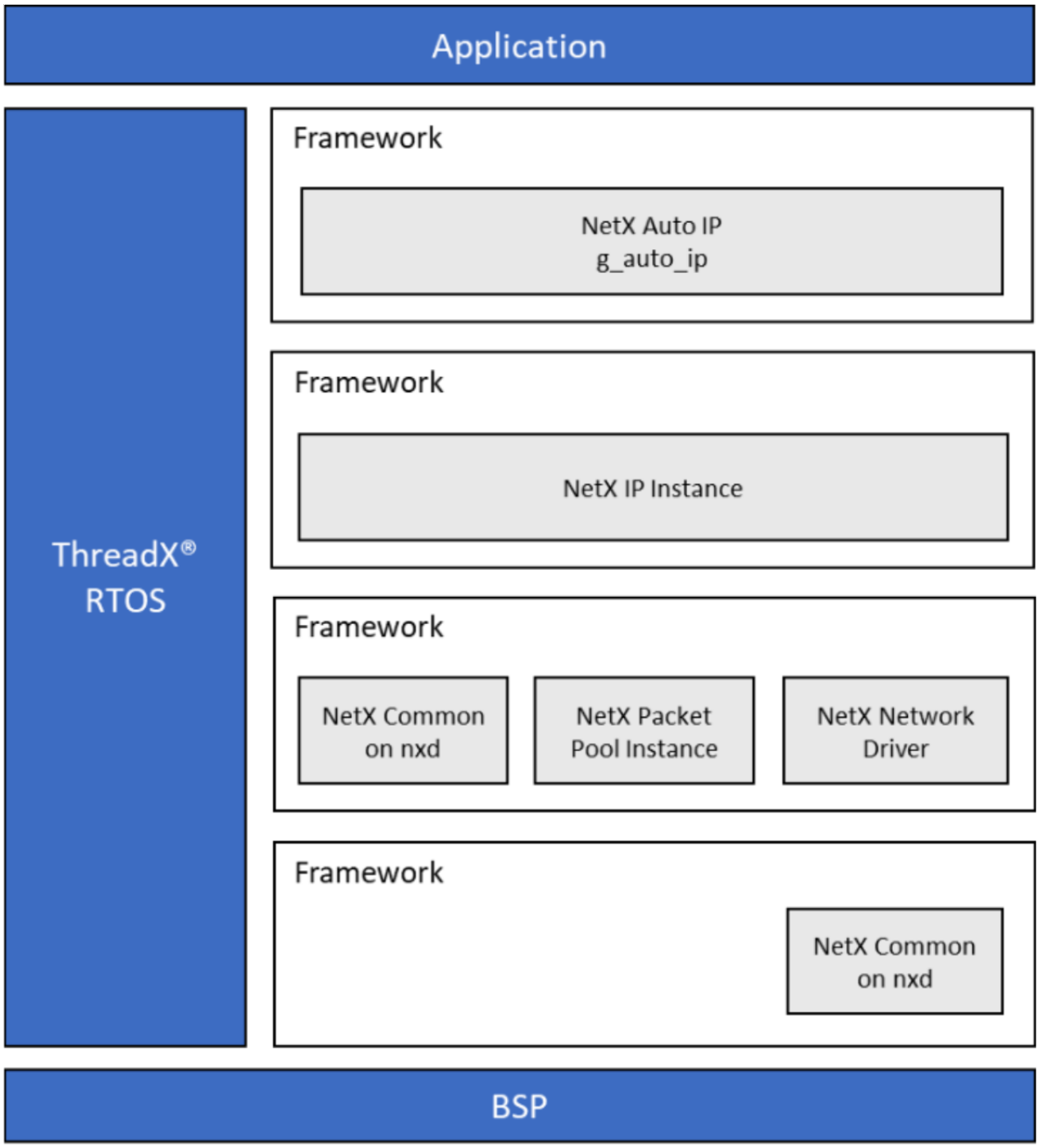


Figure 427: NetX/NetX Duo Auto IP Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo Auto IP Module in an Application for additional details.

4.3.12.2 NetX/NetX Duo Auto IP Module APIs Overview

The NetX Auto IP defines APIs for creating, deleting, getting and setting addresses. A complete list of the available APIs, an example API call and a short description of each can be found in the following

table. A table of status return values follows the API summary table.

NetX/NetX Duo Auto IP Module API Summary

Function Name	Example API Call and Description
nx_auto_ip_create	nx_auto_ip_create(&g_auto_ip0, "AutoIP 0", &g_ip0, stack_pointer, stack_size, priority); Create an Auto IP instance.
nx_auto_ip_delete	nx_auto_ip_delete(&g_auto_ip_0); Delete Auto IP instance.
nx_auto_ip_get_address	nx_auto_ip_get_address(&g_auto_ip_0, &local_address); Get current Auto IP address.
nx_auto_ip_set_interface	nx_auto_ip_set_interface(&g_auto_ip_0, interface_index); Set network interface needing an Auto IP address.
nx_auto_ip_start	nx_auto_ip_start(&g_auto_ip_0, IP_ADDRESS(0,0,0,0)); Start Auto IP processing. If the address input is NULL. NetX Auto IP randomly assigns an address in the Auto IP address range.
nx_auto_ip_stop	nx_auto_ip_stop(&g_auto_ip_0); Stop Auto IP processing.
nx_dhcp_server_stop	nx_dhcp_server_stop(&dhcp_server); Stop DHCP server processing.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful AutoIP function.
NX_AUTO_IP_ERROR	Error creating components of Auto IP instance.
NX_PTR_ERROR*	Invalid pointer input.
NX_CALLER_ERROR*	Invalid caller of this service.
NX_AUTO_IP_NO_LOCAL	No Auto IP address registered with the NetX IP instance.
NX_AUTO_IP_BAD_INTERFACE_INDEX	Invalid network interface.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.12.3 NetX/NetX Duo Auto IP Module Operational Overview

The NetX Auto IP protocol first selects a random address within the Auto IP IPv4 address range of 169.254.1.0 through 169.254.254.255. Alternatively, the application may force a starting IP address by providing it to the nx_auto_ip_startservice; this is useful in situations where an Auto IP address has been used previously.

Once an auto IP address is selected, the NetX Auto IP sends out a series of ARP probes for the selected address. An ARP probe consists of an ARP request message with the sender address set to 0.0.0.0 and the target address set to the desired Auto IP address. A series of these ARP probes are sent (the actual number is set by the ARP probes to send property of the NetX Auto IP instance); if another network node responds to this probe or sends an identical probe for the same address, a new auto IP address is randomly selected within the auto IP IPv4 address range and the probe processing repeats.

If ARP probes to send and probes are sent without any responses, the NetX Auto IP issues many ARP announcements (set by the Number of ARP announces property) for the selected address. An ARP announcement consists of an ARP request message with both the sender and target address in the ARP message set to the selected auto IP address. If another network node responds to an announced message or sends an identical announcement for the same address, a new auto IP address is randomly selected within the auto IP IPv4 address range, and the probe processing starts over. When the probe and announcement completes without any detected conflicts, the selected auto IP address is considered valid and the address is registered with the IP instance.

The NetX Auto IP registers the auto IP-generated IP address with the NetX IP instance successful probe and announcement processing. The Auto IP application can be notified of address changes using the nx_ip_address_change_notify callback in NetX, or it can use the nx_ip_status_check to determine when a valid IP address is assigned. Once a valid address is assigned, the application should stop the auto IP task using the nx_auto_ip_stop service. The address change callback notifies the application of address changes after the auto IP thread task is suspended. Possible reasons for an address changing without explicitly being done with an auto IP may be due to auto IP-address conflicts with other nodes, or a DHCP address resolution to replace the auto IP address.

NetX/NetX Duo Auto IP Module Important Operational Notes and Limitations

NetX/NetX Duo Auto IP Module Operational Notes

- The NetX DHCP Client and NetX Auto IP can both be used to ensure a host has a valid IP address. Typically, the DHCP Client attempts to contact a server. If none of the servers responds to the DHCP Client, the client is suspended and the auto IP task is started. Auto IP generally guarantees a local address even if no DHCP Server is available. The DHCP Client can try later to broadcast requests to a DHCP Server; this process, if successful, automatically overwrites the auto IP local address.
- When the IP address changes, the application is responsible for closing out existing socket connections.

NetX/NetX Duo Auto IP Module Limitations

- If the NetX DHCP is used with the auto IP, the DHCP thread created must have a higher priority than the auto IP thread.

- The NetX Auto IP does not provide a mechanism to retain previously used IP address.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.3.12.4 Including the NetX/NetX Duo Auto IP Module in an Application

This section describes how to include either or both NetX/NetX Duo Auto IP module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo Auto IP module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo Auto IP Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_auto_ip0 NetX Auto IP	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetX Auto IP**
g_auto_ip0 NetX Duo Auto IP	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo Auto IP**

When the NetX/NetX Duo Auto IP module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

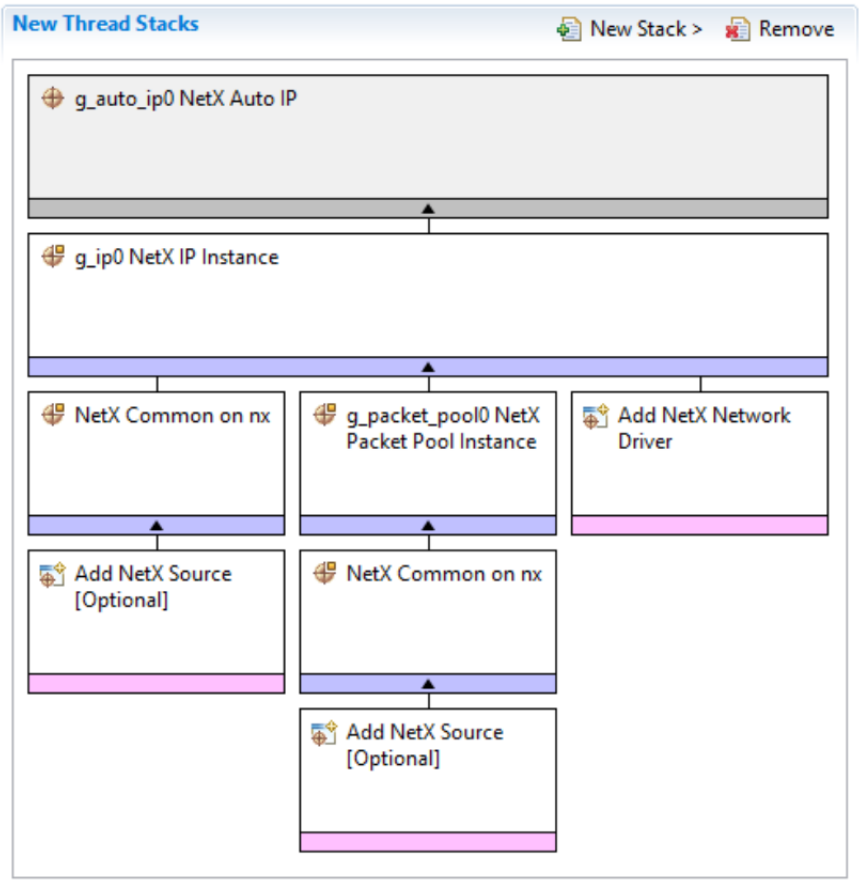


Figure 428: NetX/NetX Duo Auto IP Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.12.5 Configuring the NetX/NetX Duo Auto IP Module

The NetX/NetX Duo Auto IP module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo Auto IP Module

ISDE Property	Value	Description
Wait before sending first probe (seconds)	1	Wait before sending first probe selection.
ARP probes to send	3	ARP probes to send selection.
Minimum wait between probes (seconds)	1	Minimum wait between probes selection.
Maximum wait between probes (seconds)	2	Maximum wait between probes selection.
Maximum conflicts before increasing processing delay	10	Maximum conflicts before increasing processing delay selection.
Wait extend after maximum conflicts (seconds)	60	Wait extend after maximum conflicts selection.
Wait before announcement (seconds)	2	Wait before announcement selection.
Number of ARP announces	2	Number of ARP announces selection.
Wait between announces (seconds)	2	Wait between announces selection.
Wait between defense announces (seconds)	10	Wait between defense announces selection.
Name	g_auto_ip0	Module name.
Internal thread stack size (bytes)	2048	Internal thread stack size selection.
Internal thread priority	3	Internal thread priority selection.
Name of generated initialization function	auto_ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different addresses for the Ethernet port. The configurable properties for

the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo Auto IP Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.

ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo Auto IP Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo Auto IP Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.

MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.12.6 Using the NetX/NetX Duo Auto IP Module in an Application

In a typical application, it is assumed that an IP instance has been created and an ARP is enabled. Once this IP instance is accomplished, the typical steps in using the NetX Auto IP in an application are:

1. Allow time for the IP thread task and the network driver to get initialized (2-3 seconds) using the tx_thread_sleep API.
2. Set the address change notification with the nx_ip_address_change_notify API [Optional].
3. Start the Auto IP instance with the nx_auto_ip_start API.
4. Check for a valid address for the IP instance using either the nx_ip_status_check or nx_auto_ip_get_address API. The nx_ip_status_check API defaults to the primary address. If running Auto IP on a secondary interface, use the nx_ip_interface_status_check. Note that nx_auto_ip_get_address API works for Auto IP on either primary or secondary addresses.
5. If a valid local IP address is assigned, stop the auto IP thread task using the nx_auto_ip_stop API.

The following figure illustrates common steps in a typical operational flow diagram:

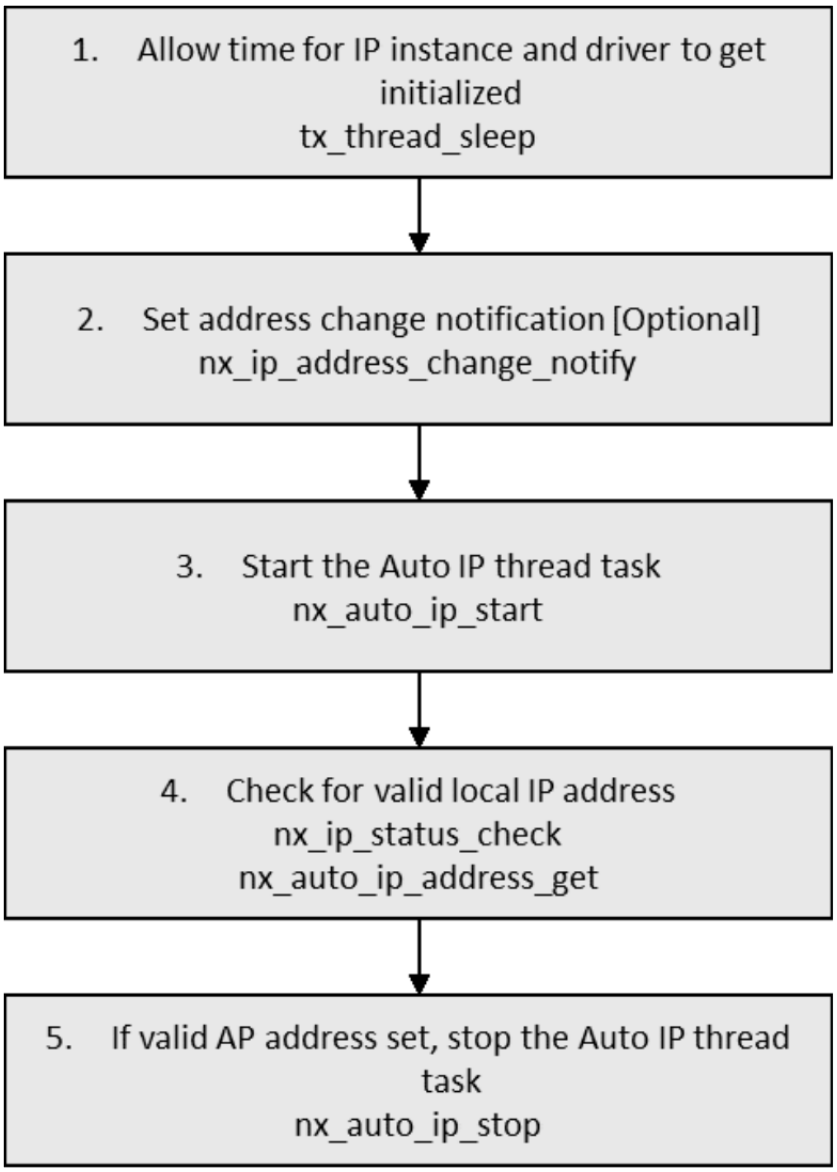


Figure 429: Flow Diagram of a Typical NetX/NetX Duo Auto IP Module Application

4.3.13 NetX/NetX Duo BSD Support

4.3.13.1 NetX/NetX Duo BSD Support Introduction

The BSD Socket API Compliancy Wrapper (NetX™ BSD) supports a subset of the basic BSD Socket API calls (with some limitations) using NetX™ services.

Note

Except for internal processing, the NetX Duo™ BSD Support module is identical in the application, set-up and running of an BSD Support session as the NetX™ BSD Support module.

Unsupported Features

BSD with DNS support has not been tested in this version of SSP.

NetX/NetX Duo BSD Support Module Features

- The NetX BSD Support module is compliant with BSD 4.3.
- Provides high-level APIs to:
    - Create and delete sockets
    - Set socket options
    - Request TCP connections and listen for connection requests
    - Send and receive packets
    - Raw packet support**
    - PPP over Ethernet support**

** NetX Duo BSD only

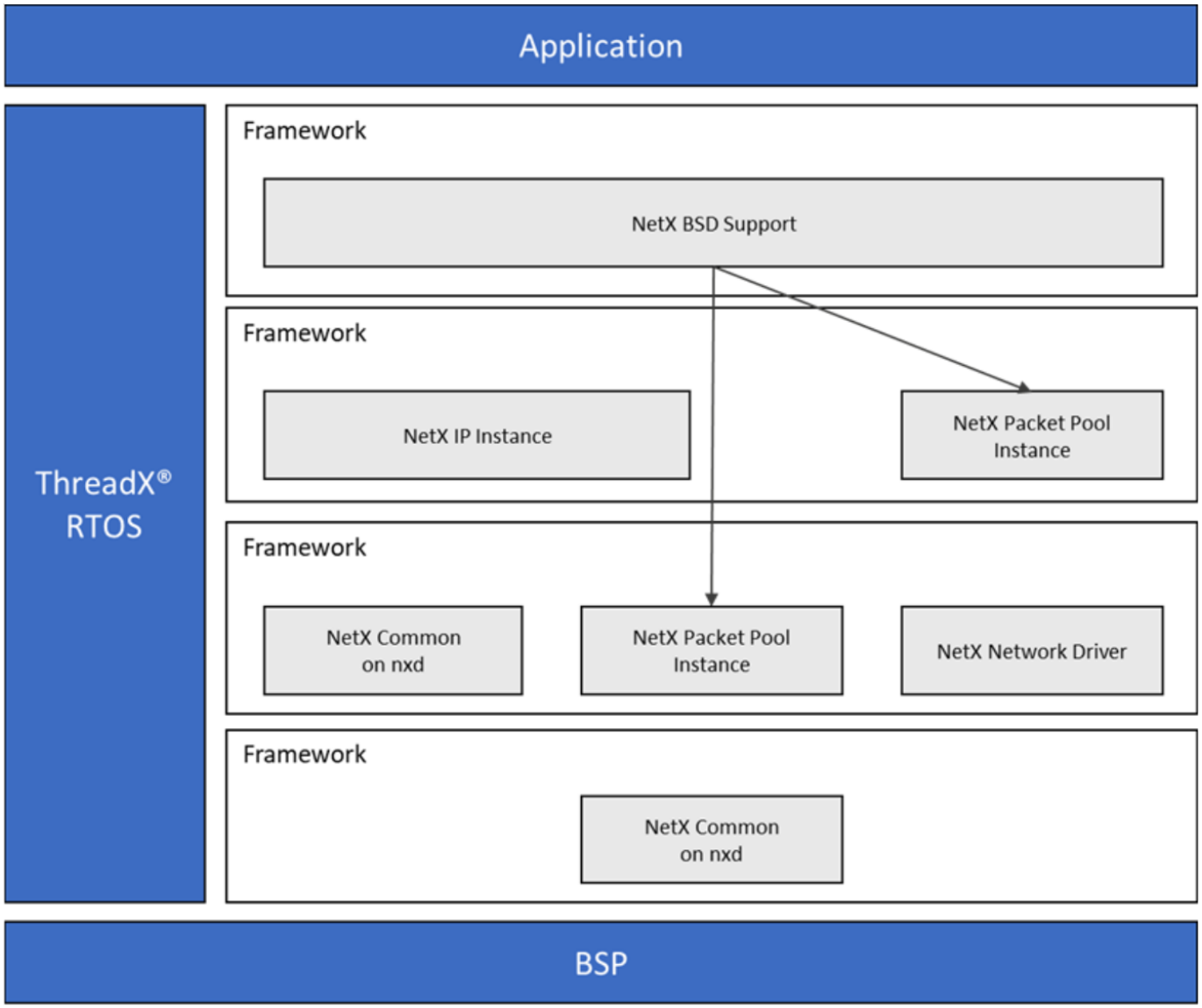


Figure 430: NetX/NetX Duo BSD Support Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options

available. See the description just after the module stack figure in the Including the NetX/NetX Duo BSD Support Module in an Application section for additional details.

4.3.13.2 NetX/NetX Duo BSD Support Module APIs Overview

The NetX BSD Support module provides the standard BSD API functions for connecting, binding, listening, sending and receiving. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. Services that are implemented by NetX and NetX Duo BSD begin with nx_, and are listed at the end of this table.

NetX/NetX Duo BSD Support Module API Summary

Function Name	Example API Call and Description
accept	accept(INT sockID, struct sockaddr *ClientAddress, INT *addressLength); TCP server socket waits to make a TCP connection.
bsd_initialize	bsd_initialize(NX_IP *default_ip, NX_PACKET_POOL *default_pool, CHAR *bsd_thread_stack_area, ULONG bsd_thread_stack_size, UINT bsd_thread_priority); Sets up BSD Support Module to use NetX and BSD services. (Called by the NetX BSD Support Module automatically.)
bind	bind(INT sockID, struct sockaddr *localAddress, INT addressLength); Bind TCP or UDP socket to a local port.
connect	connect(INT sockID, struct sockaddr *remoteAddress, INT addressLength); Connect to a TCP peer; if the remoteAddress indicates raw or UDP socket, then if the address is NULL this dis-associates the peer from this socket.
fcntl	fcntl(INT sock_ID, UINT flag_type, UINT f_options); Sets socket options for the specified socket.
freeaddrinfo	freeaddrinfo(struct addrinfo *res); Releases memory allocated by the getaddrinfo service.
getnameinfo	getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags); Converts a socket address to a corresponding host and service string.
getpeername	getpeername(INT sockID, struct sockaddr *remoteAddress, INT *addressLength); Return remote peer's IP address and port.

getsockname	getsockname(INT sockID, struct sockaddr *localAddress, INT *addressLength); Return the socket's primary IP address and source port. In NetX Duo, this would be the address as index 1 in the IP interface table.
getsockopt	getsockopt(INT sockID, INT option_level, INT option_name, VOID *option_value, INT *option_length); Return the status of the specified socket option.
getaddrinfo	getaddrinfo(const CHAR *node, const CHAR *service, const struct addrinfo *hints, struct addrinfo **res); Fills in the addrinfo for the indicated node (host) based on hints in the addrinfo input.
ioctl	ioctl(INT sockID, INT command, INT *result); Carry out the command on the specified socket. Only two options supported FIONREAD (extract number of bytes on socket queue) and FIONBIO (enable/disable non blocking as per the result flag).
inet_addr	inet_addr(const CHAR *buffer); Convert an IP address from a string buffer to a number.
inet_ntoa	inet_ntoa(struct in_addr address_to_convert); Convert an IP address to a string.
inet_aton	inet_aton(const CHAR * address_buffer_ptr, struct in_addr *addr); Converts hexadecimal characters into an ASCII IP address representation.
inet_pton	inet_pton(INT af, const CHAR *src, VOID *dst); Converts an IP address from string to numeric format.
inet_ntop	inet_ntop(INT af, const VOID *src, CHAR *dst, socklen_t size); Converts an IP address from numeric format to string presentation.
listen	listen(INT sockID, INT backlog); Bind a TCP server socket to a local port on which to listen for connection requests.
recvfrom	recvfrom(INT sockID, CHAR *buffer, INT buffersize, INT flags,struct sockaddr *fromAddr, INT *fromAddrLen); Receive up to the specified number of bytes on the specified socket (either UDP or TCP).

recv	recv(INT sockID, VOID *rcvBuffer, INT bufferLength, INT flags) Copies up to a specified number of bytes received on the socket into specified location. The given socket can be UDP or TCP, but must be in the connected state.
send	send(INT sockID, const CHAR *msg, INT msgLength, INT flags) Send the specified buffer out on the socket; the actual number of bytes sent is returned in msglength. Does not support raw sockets.
sendto	sendto(INT sockID, CHAR *msg, INT msgLength, INT flags, struct sockaddr *destAddr, INT destAddrLen); Send the specified buffer out on the socket; the actual number of bytes sent is returned in msglength. The socket must be in a connected state. Supports raw sockets (NetX Duo BSD only).
select	select(INT nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout); Check all sockets specified in the fd_set inputs to be checked for read request (incoming packets), write request (outgoing packets), and exception request input.
soc_close	soc_close(INT sockID); Close the specified socket.
socket	socket(INT protocolFamily, INT type, INT protocol); Creates and endpoint for communication and returns a file descriptor for the socket.
setsockopt	setsockopt(INT sockID, INT option_level, INT option_name, const VOID *option_value, INT option_length); Enable the input socket option on the socket.
fcntl	fcntl(INT sock_ID, UINT flag_type, UINT f_options); Sets flag options for the specified socket.
nx_bsd_raw_packet_info_extract**	nx_bsd_raw_packet_info_extract(NX_PACKET *packet_ptr, NXD_ADDRESS *nxd_address, UINT *interface_index); Extracts source IP address and interface index of the IP address in the IP interface table.

nx_bsd_socket_set_inherited_settings	nx_bsd_socket_set_inherited_settings(UINT master_sock_id, UINT secondary_sock_id); Apply the socket options of the specified master socket to the specified child secondary socket; requires NX_BSD_INHERIT_LISTENER_SOCKET_SETTINGS be defined. If it is not, this function has no effect.
nx_bsd_set_service_list	nx_bsd_set_service_list(struct NX_BSD_SERVICE_LIST *serv_list_ptr, ULONG serv_list_len); Define the service list used by getaddrinfo with the specified service list.

Note

*When inet_ntop() function is used in the application to convert IP address from ULONG to string, IP address gets reversed (From SSP 2.0.0 and later). Hence IP address has to be passed in Network byte order. The function '**htonl()**' has to be used to convert IP address into network byte order.*
For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo FTP Client. For definitions of of NetX Duo specific data types, see the*NetX Duo User Guide for the Renesas Synergy™ Platform*.

4.3.13.3 NetX/NetX Duo BSD Support Module Operational Overview

To utilize NetX and BSD services, the NetX BSD Support module automatically creates an IP instance, and creates memory space for the internal BSD thread stack. The packet pool can be the IP default packet pool (g_packet_pool0) or by clicking on **Add NetX Packet Pool (or Add NetX Duo Packet Pool)**> **New**, a separate packet pool, g_packet_pool1, will be used for BSD packet transmissions. For memory space, the defining parameters are the internal_thread_stack_sizeand internal_thread_priorityproperties of the NetX BSD Support stack element.

Before using NetX BSD services, the application creates one or more sockaddr_in instances local and peer hosts. A server application will only need to create a sock_addr for itself. (These are limited to IPv4 addressing.) For IPv6 addresses (which requires NetX Duo BSD), the application creates one or more sockaddr_in6instances. A socket of type TCP or UDP is created using the socket service; the protocol must be AF_INET for IPv4 or AF_INET6 for IPv6.

For raw packets (NetX Duo BSD only), the socket must be of type AF_PACKET.

Optionally, the application can set socket options such as non-blocking using the fnctl and ioctl services.

For TCP and UDP clients, the socket must bind a local-source port using the bind service. A value of zero for the port in the sockaddr_in the data instance will inform NetX to assign a local port. For the TCP sockets, the client socket connects to a TCP server using the connect service.

Both the UDP and TCP sockets can then send and receive data. Because the BSD is a streaming based protocol, data is delivered to and from the application in user-defined buffers. Internally, NetX sends and transmits data using packets from the packet pools transparently to the application. There is no requirement to release received packets or allocated packets that NetX was not able to send.

To close a client socket, the application calls soc_close. There is no need to unbind the socket when

closing it using soc_close.

For TCP servers, the application chooses a local port to listen on by the specified master socket using the listen service. The master socket then checks for connection requests using the select service. When one is detected, it calls the accept service and assigns a secondary socket to handle the connection. In this manner, the BSD can maintain multiple connections simultaneously.

To close a server socket, the application calls soc_close. Unlike NetX TCP sockets, there is no need to call the nx_tcp_server_socket_unlisten or nx_tcp_server_socket_unaccept API on the TCP socket in NetX BSD.

Eliminating an Internal BSD Thread

By default, BSD utilizes an internal thread to perform some of its processing. In systems with tight memory constraints, NetX BSD can be built with NX_BSD_TIMEOUT_PROCESS_IN_TIMER defined, which eliminates the internal BSD thread and alternatively uses an internal timer to perform the same processing. This eliminates the memory required for the internal BSD thread control block and stack. The timer processing is significantly increased and the BSD processing may also execute at a higher priority than needed.

To configure BSD sockets to run in the ThreadX-timer context, define NX_BSD_TIMEOUT_PROCESS_IN_TIMER in the project. If the BSD layer is configured to execute the BSD tasks in the timer context, the following properties of the BSD stack element are ignored:

- internal_thread_stack_size
- internal_thread_priority

NetX BSD with DNS Support

If NX_BSD_ENABLE_DNS is defined, NetX BSD can send DNS queries to obtain hostname or host IP information. This feature requires a NetX DNS Client instance to be previously created. The BSD link to this DNS instance is via an extern NX_DNS *_nx_dns_instance_ptr. When the BSD application calls getnameinfo with an address or getaddrinfo with a hostname, NetX BSD will call various NetX DNS Client services to obtain the host name or IP address, respectively. Refer to the *NetX DNS Client User Guide* for the Renesas Synergy Platform, available as described at the Reference section at the end of this document, for more details on setting up a DNS Client in an application.

Raw Socket Support (NetX Duo BSD only)

To use raw sockets in the NetX Duo BSD, the NetX Duo library must be compiled with NX_ENABLE_IP_RAW_PACKET_FILTERdefined in the project. By default, it is not defined. To define it, right click on the project in the **Synergy Configuration pane**> **Properties**> **Cross ARM C Compiler**> **Preprocessor** and add this to the list of defined symbols by clicking on the (+) icon.

The application must then enable raw socket processing for a previously created IP instance by calling the nx_ip_raw_packet_enable service before using NetX Duo BSD services. To create a raw socket in NetX Duo BSD, the application uses the socket service and specifies the protocol family, socket type, and protocol:

sock_1 = socket(INT protocolFamily, INT socket_type, INT protocol)

The NetX BSD supports these values for protocol Family : AF_INET for IPv4 sockets, AF_INET6 for IPv6 sockets, and AF_PACKET for raw sockets. The socket_type must be set to SOCK_RAW. The protocol is application specific.

To send and receive raw packets as well as close a raw socket, the application typically uses the same BSD services as the UDP like sendto, recvfrom, select, andsoc_close. Raw sockets do not support either accept or listen BSD services.

- By default, received IPv4 raw data includes the IPv4 header. Conversely, received IPv6 raw data does not include the IPv6 header.
- By default, when sending either raw IPv6 or IPv4 packets, the BSD wrapper layer adds the IPv6 or IPv4 header before sending the data.

The NetX Duo BSD supports additional raw socket options, includingIP_RAW_RX_NO_HEADER, IP_HDRINCL and IP_RAW_IPV6_HDRINCL. If IP_RAW_RX_NO_HEADER is set, the IPv4 header is removed so that the received data does not contain the IPv4 header, and the reported message length does not include the IPv4 header. For IPv6 sockets, by default the raw socket receive does not include the IPv6 header, equivalent to having the IP_RAW_RX_NO_HEADER option set. The application may use the setsockopt service to clear the IP_RAW_RX_NO_HEADER option. Once the IP_RAW_RX_NO_HEADEROption is cleared, the received IPv6 raw data would include the IPv6 header, and the reported message length includes the IPv6 header. This option has no effect on IPv4 or IPv6 transmitted data.

If IP_HDRINCL is set, the application includes the IPv4 header when sending data. This option has no effect on IPv6 transmission and is not defined by default. If the IP_RAW_IPV6_HDRINCL is set, the application includes the IPv6 header when sending data. This option has no effect on IPv4 transmission and is not defined by default.

IP_HDRINCLand IP_RAW_IPV6_HDRINCL have no effect on IPv4 or IPv6 reception.

Note

The BSD 4.3 Socket specification specifies that the kernel must copy the raw packet to each socket receive buffer. However, in NetX Duo BSD, if multiple sockets are created sharing the same protocol, the behavior is undefined.

Raw Packet Support for PPPoE (NetX Duo BSD only)

To enable the raw packet support for PPPoE, NX_BSD_RAW_PPPOE_SUPPORT must be defined in the project. To define it, right click on the project in the **Synergy Configuration pane**> **Properties**> **Cross ARM C Compiler**> **Preprocessor** and add this to the list of defined symbols by clicking on the (+) icon. This does not require that the NetX Duo library be rebuilt.

The following command creates a BSD socket to handle PPPoE raw packets:

sockfd = socket(AF_PACKET, SOCK_RAW, protocol);

The current BSD implementation only supports two protocol types in AF_PACKET family:

- ETHERTYPE_PPPOE_DISC: PPPoE Discovery packets. In the MAC data frame, the Discovery packets have the Ethernet frame type 0x8863.
- ETHERTYPE_PPPOE_SESS: PPP Session packets. In the MAC data frame, the Session packets have the Ethernet frame type 0x8864.

The data type sockaddr_ll (the ll stands for link layer) is used to specify parameters when sending or receiving PPPoE frames.

struct sockaddr_ll is declared as:

```
struct sockaddr_ll
```

```
{
USHORT sll_family; /* Must be AF_PACKET */
USHORT sll_protocol; /* LL frame type */
INTsll_ifindex; /* Interface Index. */
USHORT sll_hatype; /* Header type */
UCHAR sll_pkttype; /* Packet type */
UCHAR sll_halen; /* Length of address */
UCHAR sll_addr[8]; /* Physical layer address */
};
```

Note that not every field in the structure is used by sendto() or recvfrom(). See the following description on how to set up the sockaddr_ll for sending and receiving PPPoE packets.

A socket created in the AF_PACKET family can be used to send either PPPoE discovery packets or PPP session packets, regardless of the protocol specified. When transmitting a PPPoE packet, application must prepare the buffer that includes properly formatted PPPoE frame, including the MAC headers (destination MAC address, source MAC address, and frame type.) The size of the buffer includes the 14-byte Ethernet header.

In thesockaddr_ll struct, the sll_ifindex is used to indicate the physical interface to be used for sending this packet. The rest of the fields in the structure are not used. Values set to the unused fields are ignored by the BSD internal process.

The following code block illustrates how to transmit a PPPoE packet:

```
struct sockaddr_ll peer_addr;
/* Transmit the PPPoE frame using the primary network interface. */
peer_addr.sll_ifindex = 0;
n = sendto(sockfd, frame, frame_size, 0, (struct
sockaddr*)&peer_addr, sizeof(peer_addr));
```

The return value indicates the number of bytes transmitted. Since PPPoE packets are message-based, on a successful transmission, the number of bytes sent matches the size of the packet, including the 14-byte Ethernet header.

PPPoE packets can be received using recvfrom(). The receive buffer must be big enough to accommodate a message of ethernet MTU size. The received PPPoE packet includes a 14-byte ethernet header. On receiving PPPoE packets, PPPoE discovery packets can only be received by socket created with protocol ETHERTYPE_PPPOE_DISC. Similarly, PPP session packets can only be received by socket created with protocol ETHERTYPE_PPPOE_SESS. If multiple sockets are created for the same protocol type, arriving PPPoE packets are forwarded to the socket created first. If the first socket created for the protocol is closed, the next socket in the order of creation is used for receiving these packets.

After a PPPoE packet is received, the following fields in the sockaddr_ll structure are valid:

- sll_family: Set by the BSD internal to be AF_PACKET
- sll_ifindex: Indicates the interface from which the packet is received
- sll_protocol: Set to the type of packet received: ETHERTYPE_PPPOE_DISC or ETHERTYPE_PPPOE_SESS

NetX/NetX Duo BSD Support Module Important Operational Notes and Limitations

NetX/NetX Duo BSD Support Module Operational Notes

NetX BSD Build Requirements

- Add the NetX Source element to the configuration and set the Extended Notify Support property to enabled.
- Define the following property in the ThreadX Source element: TX_THREAD_EXTENSION_2 int bsd_errno; By default, all the EXTENSION macros are undefined;bsd_errno can also be defined as TX_THREAD_EXTENSION_1 or TX_THREAD_EXTENSION_0.

NetX BSD Socket Options

NetX BSD socket options can be enabled (or disabled) at run time on a per socket basis using the setsockopt service, which takes as one of its inputs option_level. There are two different settings for option_level. The first type is SOL_SOCKET for socket level options. The list of supported options are:

- SO_BROADCAST

  If set, this enables sending and receiving broadcast packets from NetX sockets. This is the default behavior for NetX Duo. All sockets have this capability.

- SO_ERROR

  Used to obtain socket status on the previous socket operation of the specified socket, using the getsockoptservice. All sockets have this capability.

- SO_KEEPALIVE

  If set, this enables the TCP Keep Alive feature. This requires the NetX Duo library to be built with NX_TCP_ENABLE_KEEPALIVE defined (setting the TCP Keepalive property to enabled in the NetX and NetX Duo Common stack element). By default, this feature is disabled.

- SO_RCVTIMEO

  This sets the wait option in seconds for receiving packets on NetX Duo BSD sockets. The default value is the NX_WAIT_FOREVER (0xFFFFFFFF) or, if non-blocking is enabled, NX_NO_WAIT (0x0).

- SO_RCVBUF

  This sets the window size of the TCP socket. The default value, NX_BSD_TCP_WINDOW, is set to 64 k for BSD TCP sockets. To set the size over 65535 requires the NetX Duo library to be built with the NX_TCP_ENABLE_WINDOW_SCALINGbe defined (setting the TCP Keepaliveproperty to enabled in the NetX and NetX Duo Common stack element).

- SO_REUSEADDR

If set, this enables multiple sockets to be mapped to one port. The typical usage is for the TCP Server socket. This is the default behavior of NetX sockets.

Note

The SO_ERROR option requires that the bsd_errno is defined. To define bsd_errno, add a NetX Source Stack below the NetX Common on the nx element. Or if using NetX Duo, add a NetX Duo Source element below the NetX Duo Common on the nxd element. Then add a ThreadX Source Stack element in the NetX Duo Source. Scroll down the list of Properties and choose one of the TX_THREAD_EXTENSION macros (0?2). Set the value as follows:

- int bsd_errno;

  This is explained in NetX BSD Build Requirements section above.

  The other type is IP_PROTO for options that are implemented at the IP layer and affect all sockets. The list of run time IP level options is shown below:

- IP_MULTICAST_TTL

  This sets the time to live for UDP sockets. The default value is NX_IP_TIME_TO_LIVE (0x80) when the socket is created. This value can be overridden by calling setsockoptwith this socket option before calling the socket service.

- IP_ADD_MEMBERSHIP

  If set, this option enables the BSD socket (applies only to UDP sockets) to join the specified IGMP group.

- IP_DROP_MEMBERSHIP

  If set, this option enables the BSD socket (applies only to UDP sockets) to leave the specified IGMP group.

The following options are only supported in NetX Duo BSD:

- IP_HDRINCL

  If this option is set, the calling application must append the IP header and optionally application headers to data being transmitted on raw IPv4 sockets created in BSD. To use this option, raw socket processing must be enabled on the IP task. See the previous section Raw Socket Support for specific details.

- IP_RAW_IPV6_HDRINCL

  If this option is set, the calling application must append an IPv6 header and optionally application headers to data being transmitted on raw IPv6 sockets created by BSD. To use this option, raw socket processing must be enabled on the IP task. See the previous section Raw Socket Support for specific details.

- IP_RAW_RX_NO_HEADER

  If cleared, the IPv6 header is included with the received data for raw IPv6 sockets created in BSD. IPv6 headers are removed by default in BSD raw IPv6 sockets, and the packet length does not include the IPv6 header. If set, the IPv4 header is removed from received data on

BSD raw sockets of type IPv4. IPv4 headers are included by default in BSD raw IPv4 sockets and packet length includes the IPv4 header. This option has no effect on either IPv4 or IPv6 transmission data. See the previous section Raw Socket Support for specific details on enabling raw packet support.

To retrieve an option setting, call getsockopt for the option name with option_level again set to SOL_SOCKET for socket level options or IP_PROTOfor IP level options.

More details on run time socket level options are available in the *NetX™ BSD 4.3 Socket API Wrapper for NetX User's Guide* for the Renesas Synergy™ Platform and *NetX Duo™ BSD 4.3 Socket API Wrapper for NetX Duo User's Guide* for the Renesas Synergy™ Platform documents available as described previously in the Introduction section.

- On some systems, there may be a conflict with definition of types in the native BSD. If this happens, include _POSIX_SOURCE among the project preprocessor definitions. This is done by right clicking at the top level of the project, **Properties**> **C/C++ Build**> **Settings**> **Cross ARM C Compiler (if using that project platform)**> **Preprocessor**.

NetX BSD Build Requirements

- Add the NetX Source element to the configuration and set the Extended Notify Support property to enabled.

- Define the following property in the ThreadX Source element:

  TX_THREAD_EXTENSION_2 int bsd_errno; By default, all the EXTENSION macros are undefined;bsd_errno can also be defined as TX_THREAD_EXTENSION_1 or TX_THREAD_EXTENSION_0.

NetX BSD Socket Options

NetX BSD socket options can be enabled (or disabled) at run time on a per socket basis using the setsockopt service, which takes as one of its inputs option_level. There are two different settings for option_level. The first type is SOL_SOCKET for socket level options. The list of supported options are:

- SO_BROADCAST

  If set, this enables sending and receiving broadcast packets from NetX sockets. This is the default behavior for NetX Duo. All sockets have this capability.

- SO_ERROR

  Used to obtain socket status on the previous socket operation of the specified socket, using the getsockoptservice. All sockets have this capability.

- SO_KEEPALIVE

  If set, this enables the TCP Keep Alive feature. This requires the NetX Duo library to be built with NX_TCP_ENABLE_KEEPALIVE defined (setting the TCP Keepalive property to enabled in the NetX and NetX Duo Common stack element). By default, this feature is disabled.

- SO_RCVTIMEO

This sets the wait option in seconds for receiving packets on NetX Duo BSD sockets. The default value is the NX_WAIT_FOREVER (0xFFFFFFFF) or, if non-blocking is enabled, NX_NO_WAIT (0x0.

- SO_RCVBUF

  This sets the window size of the TCP socket. The default value, NX_BSD_TCP_WINDOW, is set to 64 k for BSD TCP sockets. To set the size over 65535 requires the NetX Duo library to be built with the NX_TCP_ENABLE_WINDOW_SCALINGbe defined (setting the TCP Keepaliveproperty to enabled in the NetX and NetX Duo Common stack element).

- SO_REUSEADDR

  If set, this enables multiple sockets to be mapped to one port. The typical usage is for the TCP Server socket. This is the default behavior of NetX sockets.

Note

The SO_ERROR option requires that the bsd_errno is defined. To define bsd_errno, add a NetX Source Stack below the NetX Common on the nx element. Or if using NetX Duo, add a NetX Duo Source element below the NetX Duo Common on the nxd element. Then add a ThreadX Source Stack element in the NetX Duo Source. Scroll down the list of Properties and choose one of the TX_THREAD_EXTENSION macros (0?2). Set the value as follows:

- int bsd_errno;

  This is explained in NetX BSD Build Requirements section above.

  The other type is IP_PROTO for options that are implemented at the IP layer and affect all sockets. The list of run time IP level options is shown below:

- IP_MULTICAST_TTL

  This sets the time to live for UDP sockets. The default value is NX_IP_TIME_TO_LIVE (0x80) when the socket is created. This value can be overridden by calling setsockoptwith this socket option before calling the socket service.

- IP_ADD_MEMBERSHIP

  If set, this option enables the BSD socket (applies only to UDP sockets) to join the specified IGMP group.

- IP_DROP_MEMBERSHIP

  If set, this option enables the BSD socket (applies only to UDP sockets) to leave the specified IGMP group.

The following options are only supported in NetX Duo BSD:

- IP_HDRINCL

  If this option is set, the calling application must append the IP header and optionally application headers to data being transmitted on raw IPv4 sockets created in BSD. To use this option, raw socket processing must be enabled on the IP task. See the previous section Raw Socket Support for specific details.

- IP_RAW_IPV6_HDRINCL

  If this option is set, the calling application must append an IPv6 header and optionally application headers to data being transmitted on raw IPv6 sockets created by BSD. To use this option, raw socket processing must be enabled on the IP task. See the previous section Raw Socket Support for specific details.

- IP_RAW_RX_NO_HEADER

  If cleared, the IPv6 header is included with the received data for raw IPv6 sockets created in BSD. IPv6 headers are removed by default in BSD raw IPv6 sockets, and the packet length does not include the IPv6 header. If set, the IPv4 header is removed from received data on BSD raw sockets of type IPv4. IPv4 headers are included by default in BSD raw IPv4 sockets and packet length includes the IPv4 header. This option has no effect on either IPv4 or IPv6 transmission data. See the previous section Raw Socket Support for specific details on enabling raw packet support.

To retrieve an option setting, call getsockopt for the option name with option_level again set to SOL_SOCKET for socket level options or IP_PROTOfor IP level options.

More details on run time socket level options are available in the *NetX™ BSD 4.3 Socket API Wrapper for NetX User's Guide* for the Renesas Synergy™ Platform and *NetX Duo™ BSD 4.3 Socket API Wrapper for NetX Duo User's Guide* for the Renesas Synergy™ Platform documents available as described previously in the Introduction section.

- On some systems, there may be a conflict with definition of types in the native BSD. If this happens, include _POSIX_SOURCE among the project preprocessor definitions. This is done by right clicking at the top level of the project, **Properties**> **C/C++ Build**> **Settings**> **Cross ARM C Compiler (if using that project platform)**> **Preprocessor**.

NetX/NetX Duo BSD Support Module Limitations

- Only MSG_DONTWAIT and MSG_PEEK flags are supported for send, recv, sendto and recvfrom calls.
- NetX BSD socket level options are limited to:
  - SO_BROADCAST
  - SO_ERROR
  - SO_KEEPALIVE
  - SO_RCVTIMEO
  - SO_RCVBUF
  - SO_REUSEADDR
- NetX BSD IP level options are limited to:
  - IP_MULTICAST_TTL
  - IP_RAW_IPV6_HDRINCL (NetX Duo BSD only)
  - IP_ADD_MEMBERSHIP
  - IP_DROP_MEMBERSHIP
  - IP_HDRINCL(NetX Duo BSD only, raw sockets must be enabled)
  - IP_RAW_RX_NO_HEADER (NetX Duo BSD only, raw sockets must be enabled)
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.13.4 Including the NetX/NetX Duo BSD Support Module in an Application

This section describes how to include either or both the NetX and NetX Duo BSD Support module in

an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo BSD Support module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo BSD Support Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
NetX BSD Support	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetX BSD Support**
NetX Duo BSD Support	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo BSD Support**

When the NetX and/or NetX Duo BSD Support module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

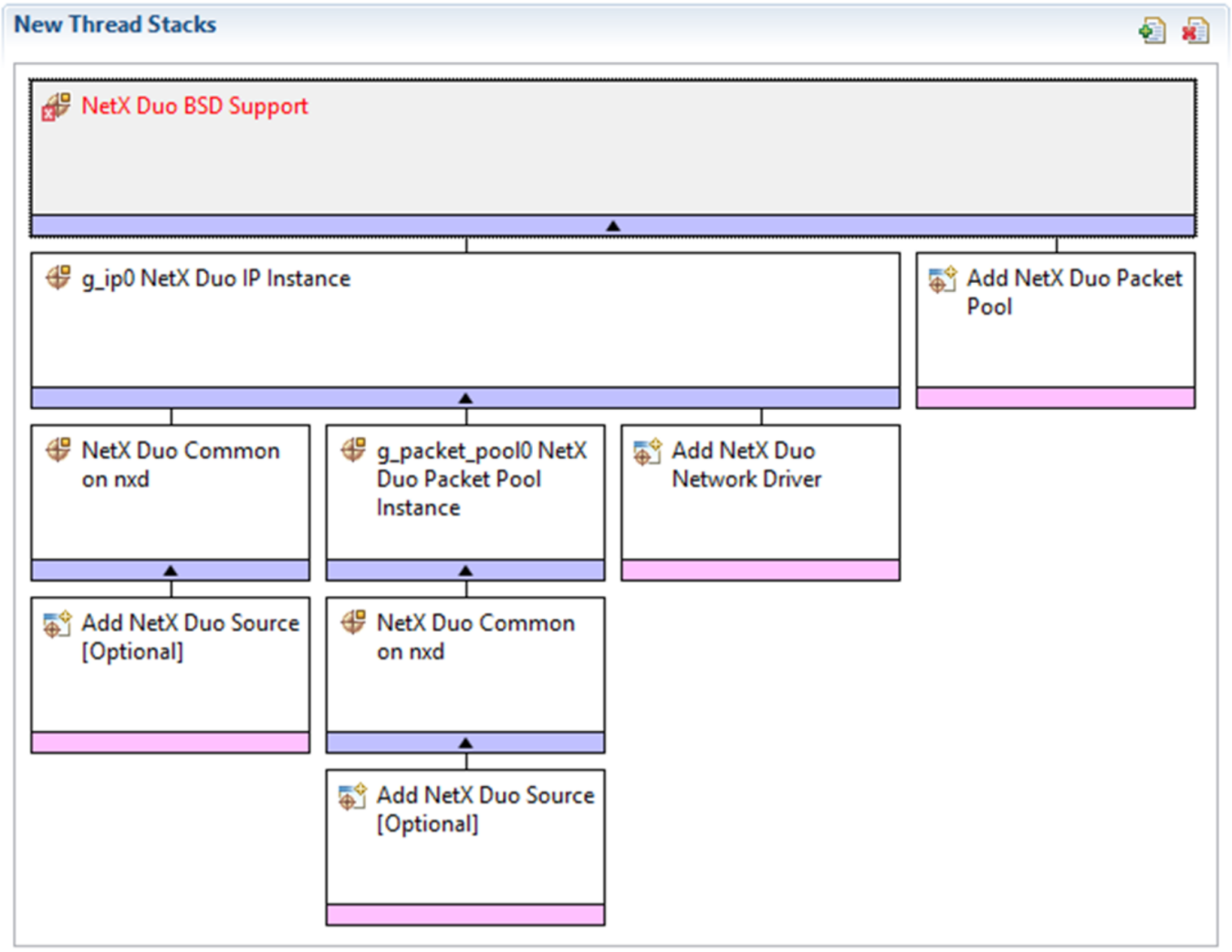


Figure 431: NetX/NetX Duo BSD Support Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.13.5 Configuring the NetX/NetX Duo BSD Support Module

The NetX/NetX Duo BSD Support module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This

approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo BSD Support Module

ISDE Property	Value	Description
NetX BSD Warning	Enable, Disable Default: Enable	NetX BSD warning selection.
Internal thread stack size(bytes)	2048	Internal thread stack size selection.
Internal thread priority	3	Internal thread priority selection.
Name of generated initialization function	nx_bsd_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC or IP Addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo BSD Support Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description

Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function.

Link status change callback	NULL	Link status change callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo BSD Support Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator **Clock** tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo BSD Support Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent

table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.13.6 Using the NetX/NetX Duo BSD Support Module in an Application

The steps in using the NetX and NetX Duo BSD Support module in a typical application are:

NetX BSD Client:

1. Poll the nx_ip_status_check API for when the IP instance has a valid IP address.
2. Create a socket using the socket API.
3. Create sockaddr_in for client and server defining IP address and port for client and server.
4. Bind to a local source port using the bind API.
5. Connect to the server using the connect API.
6. Obtain connection information using getpeername, getsockname services [Optional].
7. Send a packet to the Server using the send API.
8. Receive a packet using the recv API.
9. Close the socket using the soc_close API.

NetX BSD Server:

1. Poll the nx_ip_status_check API for when the IP instance has a valid IP address.

2. Create a master socket using the socket API.

3. Create sockaddr_in defining IP address and port for server.

4. Bind the socket to a port using the bind API.

5. Assign a local source port to listen for client requests using the listen API.

6. Check for socket requests (read, write, exception) using the select API.

7. Accept client requests and hand off the connection to a secondary socket using the accept API.

8. Receive packets using the recv API.

9. Send packets using the send API.

10. Close the socket using the soc_close API.

The following figure illustrates common steps in a typical operational flow diagram:

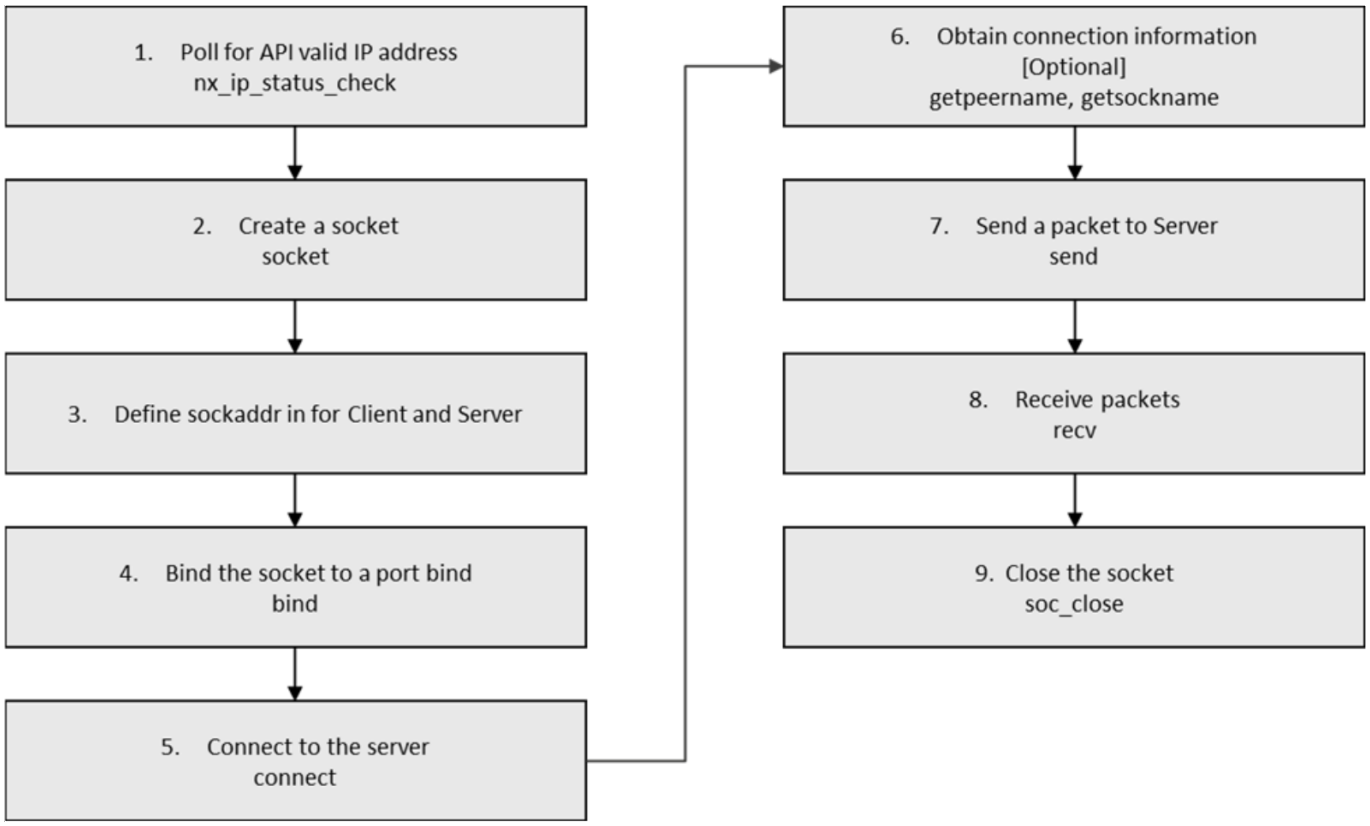


Figure 432: Flow Diagram of a Typical NetX BSD Client Module Application

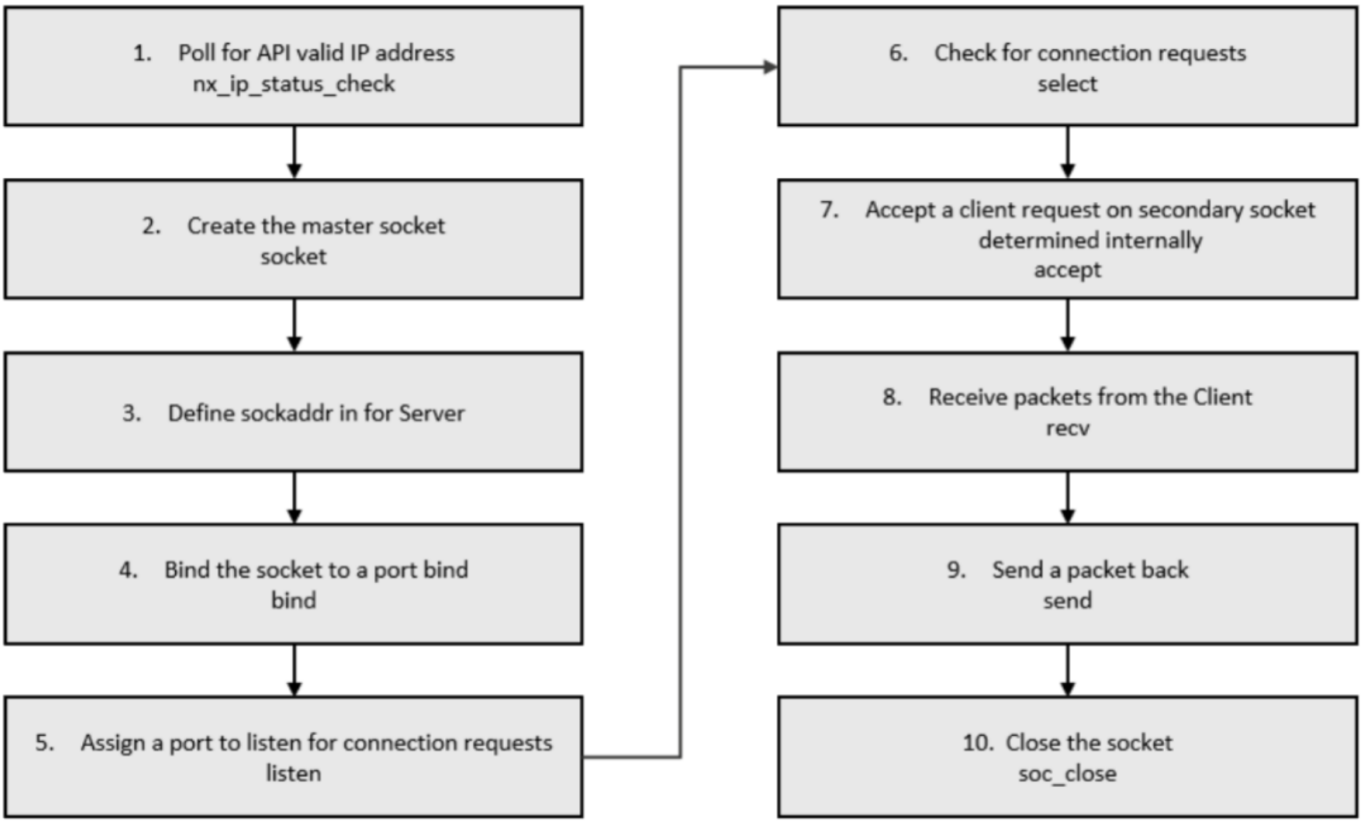


Figure 433: Flow Diagram of a Typical NetX BSD Server Module Application

NetX Duo BSD Client

1. Poll the nx_ip_status_check API for when the IP instance has a valid IP address.
2. Create a socket using the socket API.
3. Create sockaddr_in (or sockaddr_in6 for IPv6) defining IP (or IPv6) address and port for client and server.
4. Bind to a local source port using the bind API.
5. Connect to the server using the connect API.
6. Obtain connection information using getpeername, getsockname services [Optional]
7. Send a packet using the send API.
8. Receive packets using the recv and send API.
9. Close the socket using the soc_close API.

NetX Duo BSD Server

1. Poll the nx_ip_status_check API for when the IP instance has a valid IP address.
2. Create a master socket using the socket API.
3. Create sockaddr_in (or sockaddr_in6 for IPv6) defining IP (or IPv6) address and port for server Set socket options using the ioctl, setsockopt APIs
4. Bind the socket to a port using the bind API.
5. Assign a local source port to listen for client requests using the listen API.
6. Check for socket requests (read, write, exception) using the select API.
7. Accept client requests and hand off the connection to a secondary socket using the accept API.
8. Receive packets using the recv API.
9. Send packets using the send API.
10. Close the socket using the soc_close API

The following figure illustrates common steps in a typical operational flow diagram:

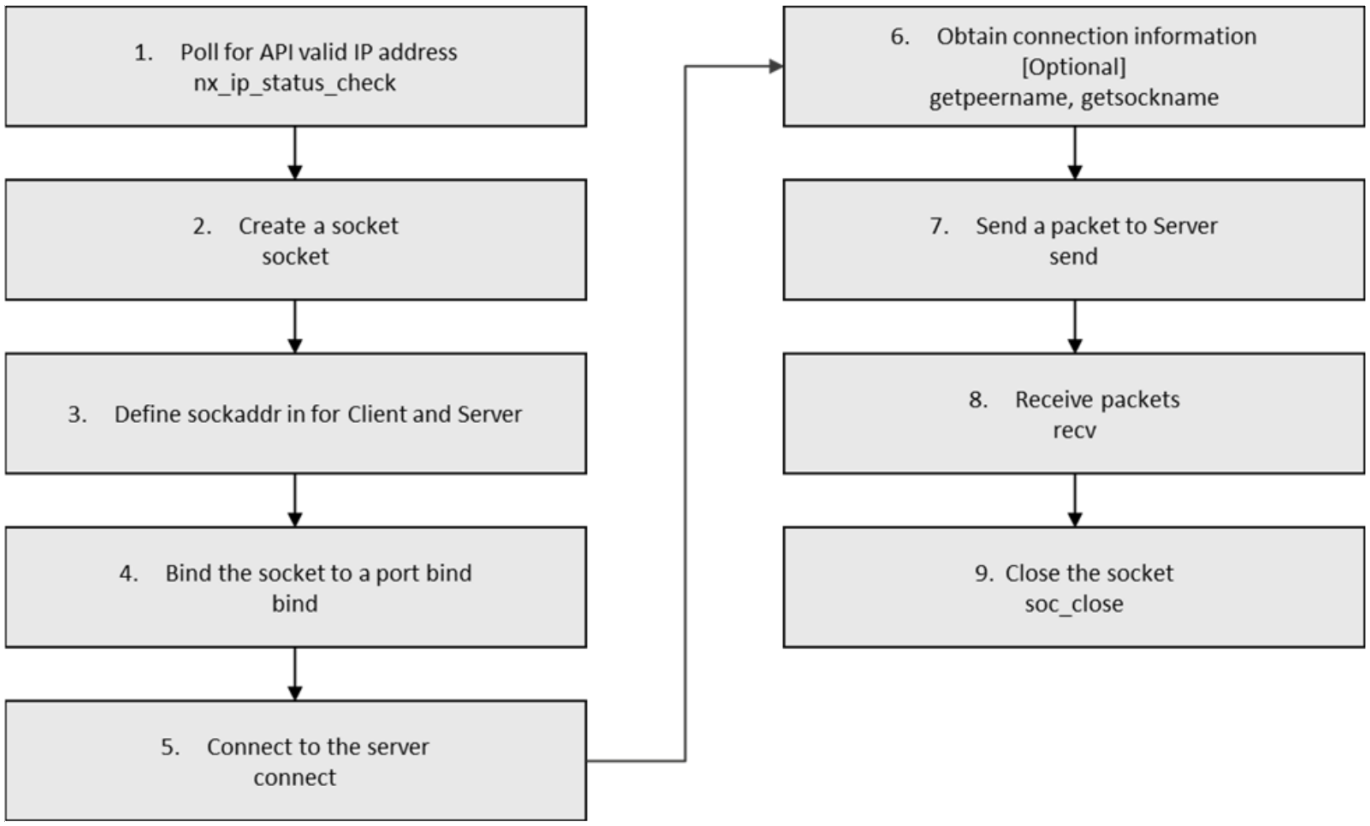


Figure 434: Flow Diagram of a Typical NetX Duo BSD Client Module Application

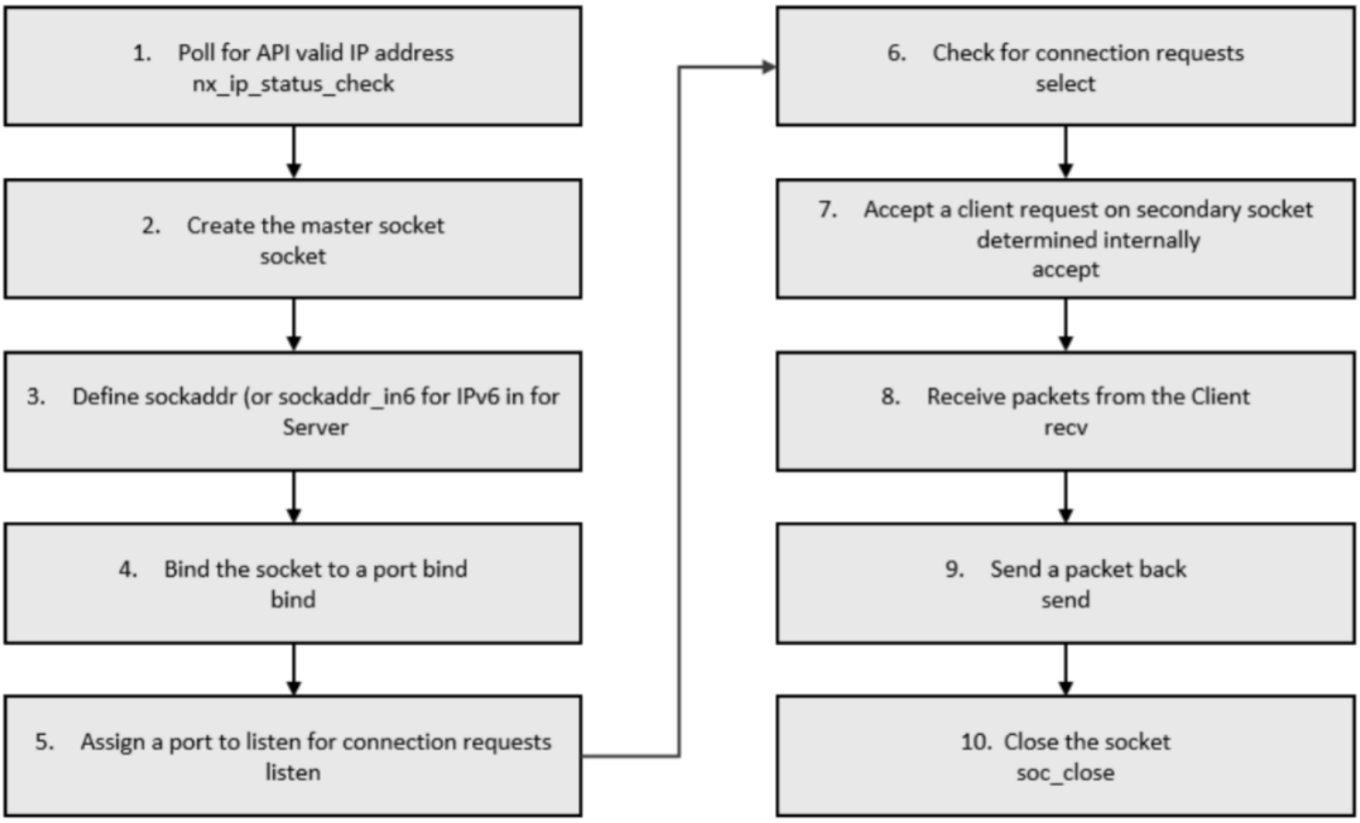


Figure 435: Flow Diagram of a Typical NetX Duo BSD Server Module Application

4.3.14 NetX/NetX Duo DHCP Client

4.3.14.1 NetX/NetX Duo DHCP Client Introduction

The Dynamic Host Configuration Protocol (DHCP) is used to obtain an IP address and network parameters. The DHCP is designed to extend the basic functionality of the BOOTP (which is limited to static address configuration) to include a completely dynamic IP address allocation through "leasing" an IP address to a client for a specified period of time. The DHCP can also be configured to allocate IP addresses in a static manner (like the BOOTP). An application's IP address is one of the supplied parameters for the NetX™ component. Supplying the IP address poses no problem if the IP address is known to the application, either statically or through the user configuration. When the application does not know or care what its IP address is, the NetX is initialized with a zero IP address; a DHCP client component added to NetX can then dynamically obtain an IP address.

In IPv6 networks, the DHCP protocol is of no use because it is limited to IPv4. Therefore, the DHCPv6 is the protocol used for dynamic global IPv6 address assignment from a DHCPv6 Server. This guide covers only the IPv4 version of DHCP, but applies to NetX™ and NetX™ Duo. A note will clearly identify where there are any differences in use between NetX and NetX Duo. To simplify wording in this document, NetX DHCPv4 will be used to stand for NetX and NetX Duo DHCP for IPv4.

NetX/NetX Duo DHCP Client Module Features

- The NetX/NetX Duo DHCP Client module is compliant with RFC2132, RFC2131 and related RFCs.
- The module provides high-level APIs to:
    - Create and delete a DHCP client instance
    - Start, stop, and reinitialize the DHCP client (to restart the DHCP client protocol)
    - Request a specific IP Address from the server
    - Specify the network interface to run the DHCP client on
    - Supply an application-created packet pool to the DHCP client

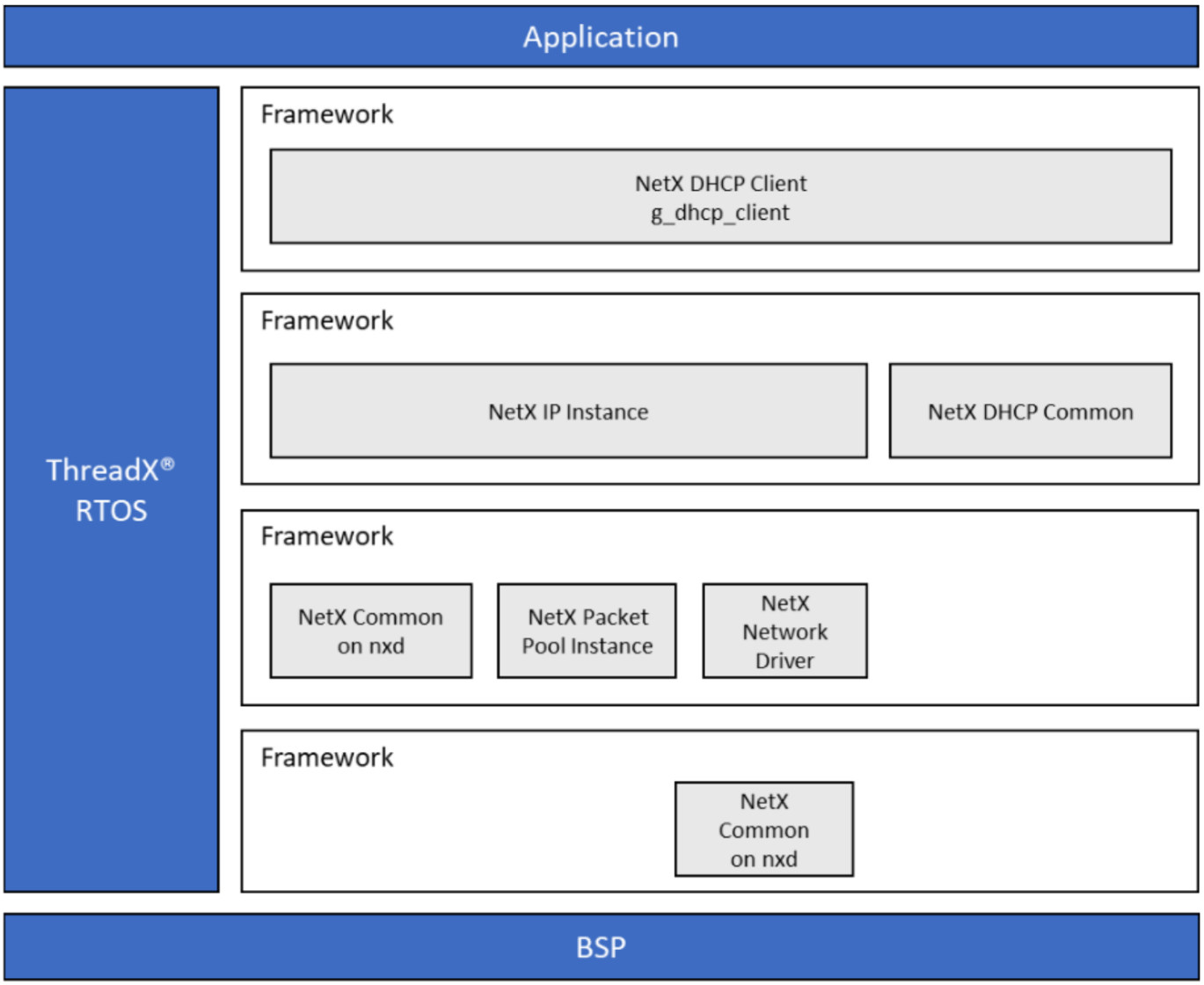


Figure 436: NetX/NetX Duo DHCP Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo DHCP Client Module in an Application for additional details.

4.3.14.2 NetX/NetX Duo DHCP Client Module APIs Overview

The NetX/NetX Duo DHCP Client module defines APIs for creating and starting the DHCP client. Internally, the DHCP client handles all communication with the DHCP server to obtain an IP address. A list of the key API functions, an example API function call and a short description of each can be found in the following table. Additional function calls are described in the NetX DHCP Client User's Manual as describe in the note below the table. A table of status return values follows the API summary table.

NetX/NetX Duo DHCP Client Module API Summary

Function Name	Example API Call and Description

nx_dhcp_create	nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP"); Create a DHCP instance.
nx_dhcp_clear_broadcast_flag	nx_dhcp_clear_broadcast_flag(&my_dhcp, NX_TRUE); Clear broadcast flag on Client messages.
nx_dhcp_delete	nx_dhcp_delete(&my_dhcp); Delete a DHCP instance.
nx_dhcp_decline	nx_dhcp_decline(&my_dhcp); Send Decline message to server.
nx_dhcp_force_renew	nx_dhcp_force_renew(&my_dhcp); Handle Server force renew message.

nx_dhcp_packet_pool_set	nx_packet_pool_create(&dhcp_pool, "DHCP Client Packet Pool",
	NX_DHCP_PACKET_PAYLOAD, pointer, (15 * NX_DHCP_PACKET_PAYLOAD));
	nx_dhcp_create(&dhcp_0, &ip_0, "janetsdhcp1");
	nx_dhcp_packet_pool_set(&my_dhcp, packet_pool_ptr);
	Set the DHCP Client packet pool. By default, the DHCP Client creates its own packet pool.

nx_dhcp_release	nx_dhcp_release(&my_dhcp); Send Release message to server.
nx_dhcp_reinitialize	nx_dhcp_reinitialize(&my_dhcp); Clear DHCP client network parameters and clear IP address and gateway registered with the IP instance.
nx_dhcp_request_client_ip	nx_dhcp_request_client_ip(&my_dhcp, IP(192,168,0,6), NX_TRUE); Request a specific IP address.
nx_dhcp_send_request	nx_dhcp_send_request(&my_dhcp, NX_DHCP_TYPE_INFORMREQUEST); Send DHCP message to server (only INFORM_REQUEST is allowed).
nx_dhcp_server_address_get	nx_dhcp_server_address_get(&dhcp_0, &server_address); Retrieve DHCP Client's DHCP server address*.*
nx_dhcp_set_interface_index	nx_dhcp_set_interface_index(&my_dhcp, 1); Specify the network interface to run DHCP Client.
nx_dhcp_start	nx_dhcp_start(&my_dhcp); Start DHCP processing.
nx_dhcp_state_change_notify	nx_dhcp_state_change_notify(&my_dhcp, my_state_change); Notify application of DHCP state change.

nx_dhcp_stop	nx_dhcp_stop(&my_dhcp); Stop DHCP processing.
nx_dhcp_user_option_retrieve	nx_dhcp_user_option_retrieve(&my_dhcp,NX_DHCP_OPTION_DNS_SVR,dns_ip_string, &size); Retrieve the specified DHCP option*.*
nx_dhcp_user_option_convert	nx_dhcp_user_option_convert(dns_ip_string); Convert four bytes to ULONG.
The following services require that *Persistent client state* be enabled	
nx_dhcp_suspend	nx_dhcp_suspend(&g_dhcp_client0); Suspend the DHCP Client thread.
nx_dhcp_resume	nx_dhcp_resume (&g_dhcp_client0); Resume the DHCP Client thread.
nx_dhcp_client_update_time_remaining	nx_dhcp_client_update_time_remaining(*g_dhcp_client0, 1000) This updates the time remaining on the IP lease by the input time in timer ticks, such as the time interval while the DHCP Client thread was suspended.
nx_dhcp_client_create_record	nx_dhcp_client_create_record(&g_dhcp_client0) This fills in a client record structure associated with the DHCP Client based on Client lease data.
nx_dhcp_client_restore_record	nx_dhcp_client_restore_record(&g_dhcp_client, client_record_ptr, time_elapsed)The Client record points to data to restore to the DHCP Client itself, and time elapsed is subtracted from the DHCP Client time remaining on its lease.

Note

For details on operation and definitions for additional API functions, the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual available in the Azure RTOS Component Documents for Renesas Synergy" zip file located at the bottom of the Renesas Synergy SSP web page here: https://www.renesas.com/en-us/products/synergy/software/ssp.html.

Status Return Values

Name	Description
NX_SUCCESS	Successful API call.
NX_PTR_ERROR*	Invalid pointer input.
NX_THREADS_ONLY_CALLER_CHECKING*	Invalid caller of this service.
NX_INVALID_INTERFACE	NetX is not enabled on the input interface
NX_NOT_ENABLED	Not enabled to set the DHCP Client packet pool.
NX_DHCP_NOT_STARTED	DHCP Client not started.

NX_DHCP_NOT_BOUND	The IP address has not been leased so the current operation is not allowed.
NX_DHCP_INVALID_MESSAGE	Illegal message type to send.
NX_DHCP_BAD_INTERFACE_INDEX*	An invalid network interface supplied.
NX_DHCP_UNKNOWN_OPTION	Unknown DHCP option to extract from DHCP server response.
NX_DHCP_INVALID_IP_REQUEST*	Invalid address for the DHCP Client to request.
NX_DHCP_INVALID_PAYLOAD	Packet pool for the DHCP Client has insufficient payload.
NX_DHCP_ALREADY_STARTED	DHCP Client thread task has already started.
NX_DHCP_PARSE_ERROR	Unable to parse requested option from Server response.
NX_DHCP_DEST_TO_SMALL	Supplied buffer too small to hold the requested option data for user requesting option data.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.14.3 NetX/NetX Duo DHCP Client Module Operational Overview

The DHCP Client module handles all the details in obtaining an IP address, registering it with the IP instance, and renewing the IP address lease before the lease expires.

A NetX IP instance is created; it has a zero IP address and is enabled for User Datagram Protocol (UDP) and the Address Resolution Protocol (ARP), respectively. The Reverse ARP (RARP) should not be enabled; a DHCP Client is then created. Its creation creates an UDP socket for sending and receiving DHCP messages. By default, the DHCP Client creates its own packet pool based on the settings *Minimum packet payload size* and *Number of packets in packet pool* (see the following table). The *Minimum Client packet payload size* must be large enough to include DHCP data, IP, UDP headers, and the physical frame header.

- For Ethernet networks, this minimum payload is 592 bytes, which is the default setting of *Minimum Client packet payload size.*
- For other network types (such as Wi-Fi), the frame-header size is larger, and minimum size must be increased correspondingly.

When the packet pool is created, the DHCP Client verifies that the packet payload is not less than the minimum required payload size.

The DHCP Client can request a specific IP address using the nx_dhcp_request_client_ip service and supply a non-zero IP address before starting the DHCP Client. Normally, the request is useful for a device previously assigned an IP address that wishes to keep the same IP address. Note: the server is not obligated to accommodate this request.

When the DHCP Client is started, it binds the socket to a DHCP port (by default 68) and begins sending and receiving packets through that socket. When the client is assigned an IP address, it automatically registers the IP address with NetX. The server supplies the network mask and network gateway, and the DHCP Client module updates NetX with that information.

When the server assigns the Client an IP address, it may also supply other network information, such as the DNS server and the NTP server. The application can obtain those values using the nx_dhcp_user_option_retrieve service.

The DHCP Client keeps track of the time remaining on the IP lease. It automatically sends Renew requests to the DHCP Server when time to renew. If the server is no longer on the network, or is otherwise not responding, the client sends broadcast requests to any DHCP Server on the network. If the lease expires without a renewal or rebinding, the client is returned to the NX_DHCP_STATE_INIT state. The device may continue to use the IP address. If a DHCP Server is later available, and the device is able to request an IP address, it must no longer use the old IP address.

In busy networks, a DHCP Client socket queue can fill up with non-specific DHCP broadcast packets intended for other DHCP Client hosts. If the DHCP Client socket receive-queue fills up, any packets intended for the device may get dropped. To avoid this problem, the DHCP Client continually clears non-specific broadcast packets from the socket.

The DHCP client module can register a callback to add options to the DHCP protocol using the nx_dhcp_user_option_add_callback_set API function. The DHCP option 60 (Vendor Class Identifier), and DHCP option 61 (Client Identifier), as well as other options, can be added for DHCP request using the registered callback. The DHCP module can chain multiple options and the options must fit in the normal payload. The Synergy configurator provides a default DHCP Option 60 addition callback. To enable the default callback, the **DHCP Option addition** and **DHCP Option addition function** properties must be set to Enable in the configurator.

NetX/NetX Duo DHCP Client Module Important Operational Notes and Limitations

NetX/NetX Duo DHCP Client Module Operational Notes

Instead of the DHCP Client module creating the packet pool, the developer may prefer to supply a previously created packet pool. To do so, enable the *Use application packet pool* option, then use the nx_dhcp_packet_pool_set service to set the DHCP Client's packet pool.

The DHCP Client verifies that the packet payload is not less than the minimum required packet size.

The IP address offered to the client should be tested for 'uniqueness' on the local network, since the DHCP protocol does not require the server to check. To configure the DHCP Client to check, enable the *Send ARP probe* option.

The DHCP Client sends a series of ARP "probes" with its assigned IP address out on the network. If any host responds to these ARP requests/probes, the DHCP Client automatically sends a DECLINE message to the server, and restarts the DHCP protocol to request another IP address. Otherwise, the DHCP Client proceeds to the bound state. The states of the client in the DHCP protocol are:

NX_DHCP_STATE_NOT_STARTED

NX_DHCP_STATE_INIT

NX_DHCP_STATE_SELECTING NX_DHCP_STATE_REQUESTING

NX_DHCP_STATE_BOUND

NX_DHCP_STATE_RENEWING

NX_DHCP_STATE_REBINDING

Note

If ARP probe is enabled, the NetX DHCP Client enters a temporary state called NX_DHCP_STATE_ADDRESS_PROBING before the NX_DHCP_STATE_BOUND state.*

The application can detect if the DHCP Client has completed (has an IP address) in a couple of ways. First, it can call the nx_ip_status_check service with the NX_IP_ADDRESS_RESOLVED option. Alternatively, it can use the _nx_dhcp_state_change_notify service which notifies the application when the DHCP Client state changes. When the DHCP Client reaches the bound state, (state == NX_DHCP_STATE_BOUND) it has a valid IP address.

If there is a need to stop the DHCP Client thread task, call the nx_dhcp_stop service. To restart the Client, first call the nx_dhcp_reinitialize service to clear the DHCP Client data and also clear network parameters registered with NetX. Then, the DHCP Client is restarted with the nx_dhcp_start call.

NetX/NetX Duo DHCP Client Module Limitations

- The DHCP Client does not support the INFORM_REQUEST message. The application can send this message out using the nx_dhcp_send_request service, but the data from the Server is not extracted and saved to the DHCP Client.
- The options supported _nx_dhcp_user_option_retrieve are limited to the following:
  - NX_DHCP_OPTION_SUBNET_MASK
  - NX_DHCP_OPTION_TIME_OFFSET
  - NX_DHCP_OPTION_GATEWAYS
  - NX_DHCP_OPTION_TIMESVR
  - NX_DHCP_OPTION_DNS_SVR
  - NX_DHCP_OPTION_NTP_SVR
  - NX_DHCP_OPTION_DHCP_LEASE
  - NX_DHCP_OPTION_DHCP_SERVER
  - NX_DHCP_OPTION_RENEWAL
  - NX_DHCP_OPTION_REBIND
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.14.4 Including the NetX/NetX Duo DHCP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo DHCP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo DHCP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo DHCP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_dhcp_client0 NetX DHCP Client	Threads	**New Stack> X-Ware> NetX> Protocols> NetX DHCP Client**
g_dhcp_client0 NetX Duo DHCP IPv4 Client	Threads	**New Stack> X-Ware> NetX Duo> Protocols> NetX Duo DHCP IPv4 Client**

When the NetX and/or NetX Duo DHCP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

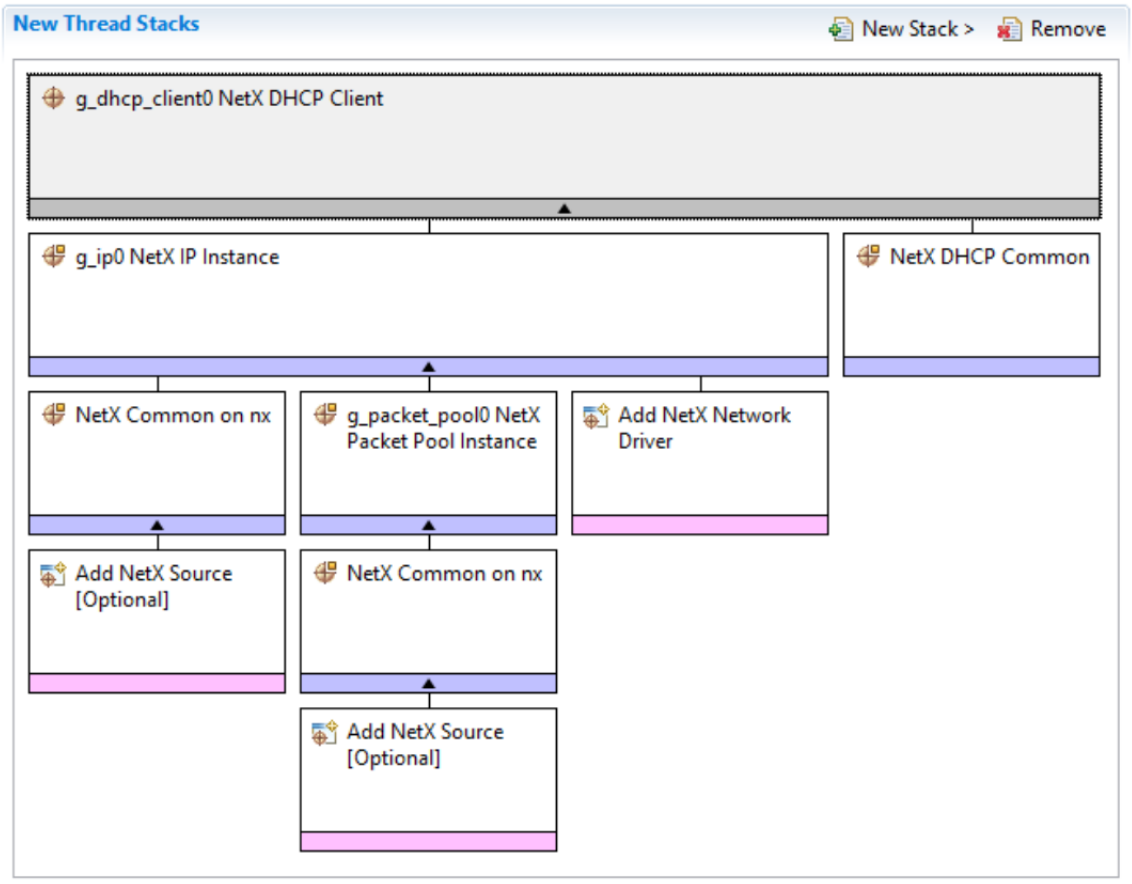


Figure 437: NetX/NetX Duo DHCP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.14.5 Configuring the NetX/NetX Duo DHCP Client Module

The NetX/NetX Duo DHCP Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo DHCP Client Module

ISDE Property	Value	Description
Internal thread priority	3	Internal thread priority selection.
Internal thread stack size (bytes)	NetX Default: 2048 NetX Duo Default: 4096	Internal thread stack size (bytes) selection.
Timeout between DHCP messages processed (seconds)	1	Timeout between DHCP messages processed (seconds) selection.
Use BOOTP	Enable, Disable Default: Disable	Use BOOTP selection.
Send ARP probe	Enable, Disable Default: Disable	Send ARP probe selection.
ARP probe wait time (seconds)	1	ARP probe wait time selection.
Minimum ARP probe wait time (seconds)	1	Minimum ARP probe wait time selection.
Minimum ARP probe wait time (seconds)	2	Minimum ARP probe wait time selection.
ARP probe count	2	ARP probe count selection.

Maximum retransmission timeout (seconds)	64	Maximum retransmission timeout (seconds) selection.
Minimum renew timeout (seconds)	60	Minimum renew timeout (seconds) selection.
Minimum retransmission timeout (seconds)	4	Minimum retransmission timeout (seconds) selection.
Client packet payload size (bytes)	592	Client packet payload size (bytes) selection.
Number of packets in internal packet pool	5	Number of packets in internal packet pool selection.
Persistent client state	Enable, Disable Default: Disable	Persistent client state selection.
Use application packet pool	Enable, Disable Default: Disable	Use application packet pool selection.
Maximum message size support	Enable, Disable Default: Disable	Maximum message size support selection.
DHCP options buffer size (bytes)	312	DHCP options buffer size (bytes) selection.
Maximum DHCP client state record on an interface	1	Maximum DHCP client state record on an interface selection.
Wait before restarting the configuration process (seconds)	10	Wait before restarting the configuration process selection.
Name	g_dhcp_client0	Module name.
Name of generated initialization function	dhcp_client_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.
*DHCP Option addition	Enable, Disable Default: Enable	Enable or Disable feature to add DHCP Option to DHCP message.
*DHCP Option addition function	Enable, Disable Default: Enable	Enable or Disable Option Addition function.
*Name of the DHCP option addition function	dhcp_user_option_add_client0	Name for the option add function provided by the user.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Settings available in NetX implementation only.*

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different Ethernet interface pins and resets. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo DHCP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.

UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.
Link status change callback	Default: NULL	Name of user defined callback function if needed- otherwise set as NULL.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo DHCP Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragmentation option selection.
Time to live	128	Time to live selection.
Packet Queue depth	5	Packet queue depth selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo DHCP Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo DHCP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence

ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.14.6 Using the NetX/NetX Duo DHCP Client Module in an Application

The following example assumes a system that is already established with a working and enabled IP, ARP and UDP, and the link is running. Additionally, set the DHCP features for the DHCP Client (request specific IP address, clear the broadcast flag, set the interface on which DHCP Client runs, or set callback to configure the user options) before starting the DHCP Client. [Optional]

The steps in using the NetX/NetX Duo DHCP Client module in a typical application are:

1. Set the DHCP features for the DHCP Client (request specific IP address, clear the broadcast flag, set the interface on which DHCP Client runs) before starting the DHCP Client. [Optional]
2. Start the DHCP using the nx_dhcp_start API.
3. Wait for IP Address resolution by calling nx_ip_status_check (a NetX library service call) or check for the bound state in the DHCP Client state-change callback function.

4. A valid IP Address is now on lease and the application can start using NetX services for sending and receiving packets.
5. The DHCP Client will automatically request IP lease renewal based on the time remaining on the IP lease (as long as the DHCP Client thread task is still running). [Optional]
6. To stop the DHCP Client thread task, call the nx_dhcp_stop API.
7. To restart the DHCP Client, call the nx_dhcp_reinitialize API and then call the nx_dhcp_start API. [Optional]
8. Add or modify the existing DHCP Client settings. [Optional]

The following figure illustrates common steps in a typical operational flow diagram:

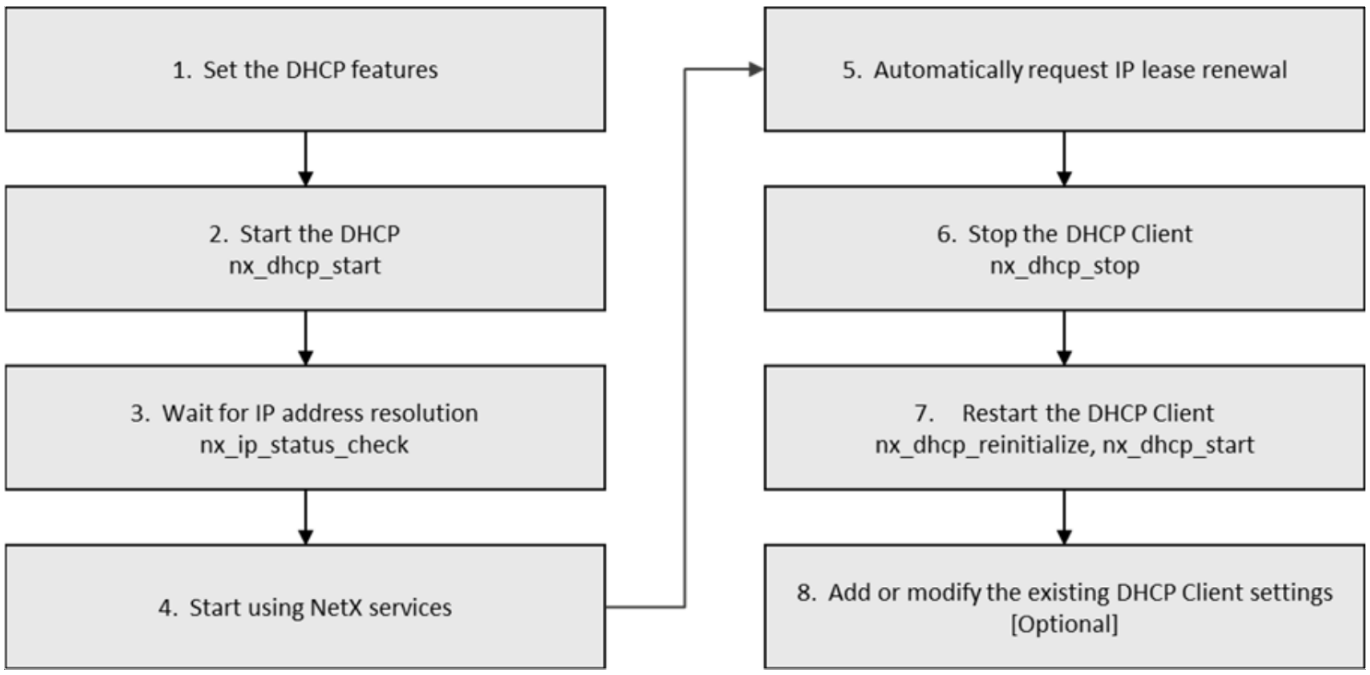


Figure 438: Flow Diagram of a Typical NetX/NetX Duo DHCP Client Module Application

4.3.15 NetX/NetX Duo DHCP Server

4.3.15.1 NetX/NetX Duo DHCP Server Introduction

The Dynamic Host Configuration Protocol (DHCP) is designed to completely automate DHCP Server allocation and dynamic IP address allocation through leasing an IP address to a client for a specified time.

In IPv6 networks, the DHCP protocol is of no use because it is limited to IPv4. Therefore, the DHCPv6 is the protocol used for dynamic global IPv6 address assignment from a DHCPv6 Server. This guide covers only the IPv4 version of DHCP, but applies to NetX™ and NetX™ Duo. A note will clearly identify where there are any differences in use between NetX and NetX Duo. To simplify wording in this document, NetX DHCPv4 will be used to stand for NetX and NetX Duo DHCP for IPv4.

NetX/NetX Duo DHCP Server Module Features

- The NetX DHCP is compliant with RFC2132, RFC2131 and related RFCs.
- Provides high-level APIs for:
    - Creating and deleting a DHCPv4 Server instance
    - Setting network parameters for DHCPv4 Server messages to the client
    - Creating a pool of assignable IP addresses
    - Starting and stopping the DHCP Server task thread

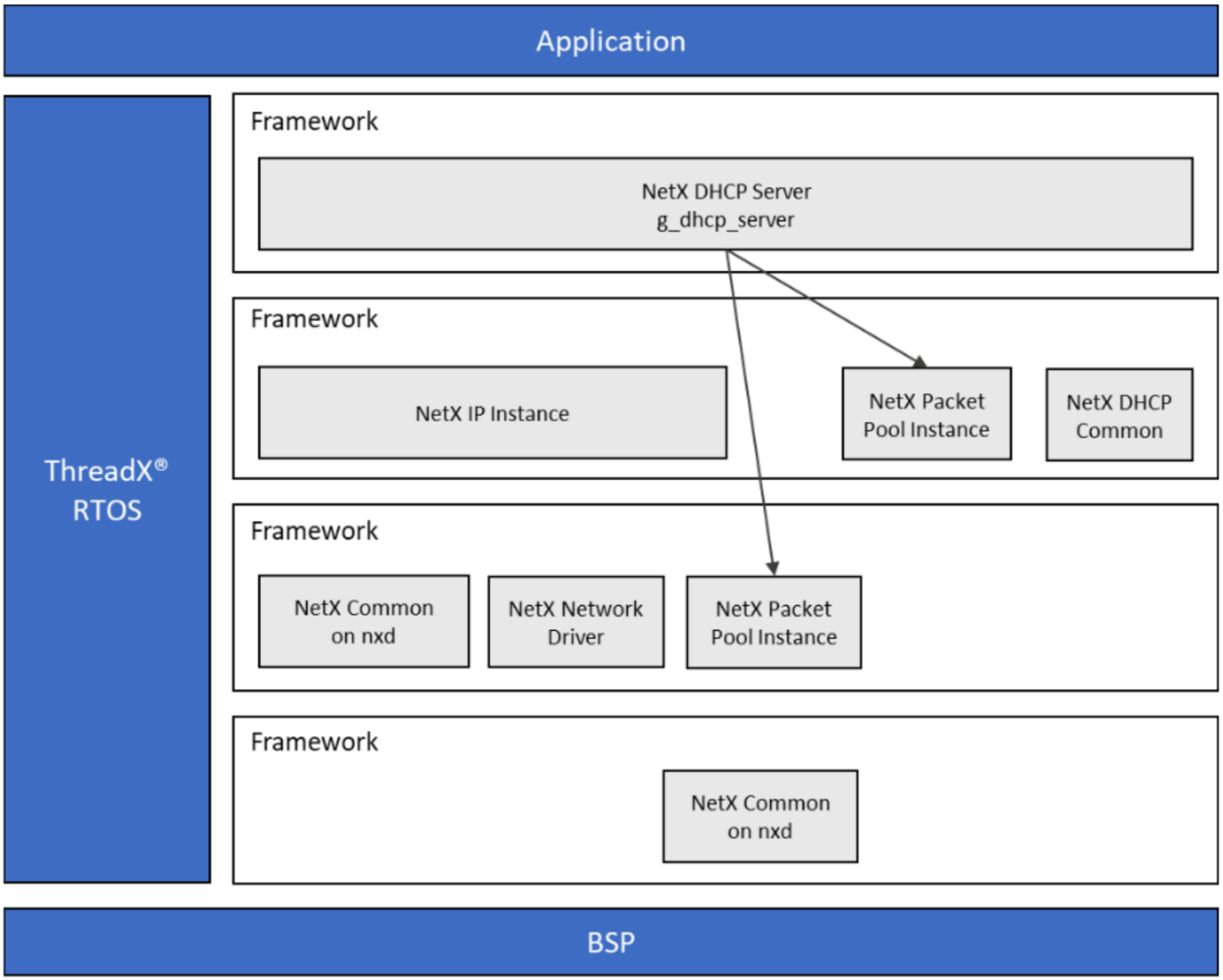


Figure 439: NetX/NetX Duo DHCP Server Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo DHCP Server Module in an Application for additional details.

4.3.15.2 NetX/NetX Duo DHCP Server Module APIs Overview

The NetX DHCP Server defines APIs for creating, deleting, removing, starting, and stopping the server, for creating the pool of assignable IP addresses, and for setting up the network information for the client. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary

table.

NetX/NetX Duo DHCP Server Module API Summary

Function Name	Example API Call and Description
nx_dhcp_server_create	nx_dhcp_server_create(&dhcp_server, &server_ip, pointer, DEMO_SERVER_STACK_SIZE, SERVER_IP_ADDRESS_LIST, "DHCP server", &server_pool); Create a DHCP Server instance.
nx_dhcp_create_server_ip_address_list	nx_dhcp_create_server_ip_list (&dhcp_server, iface_index, START_IP_ADDRESS_LIST, END_IP_ADDRESS_LIST, &addresses_added); Create pool of available IP addresses to assign to DHCP Clients on the specified network index.
nx_dhcp_clear_client_record	nx_dhcp_clear_client_record (&dhcp_server, &dhcp_client_ptr); Remove Client record in the Server database.
nx_dhcp_set_interface_network_parameters	nx_dhcp_set_interface_network_parameters(&dhcp_server, iface_index, NX_DHCP_SUBNET_MASK, NX_DHCP_DEFAULT_GATEWAY, NX_DHCP_DNS_SERVER); Set DHCP options for adding critical network parameters on specified interface in messages to Clients.
nx_dhcp_server_delete	nx_dhcp_server_delete(&dhcp_server); Delete a DHCP Server instance.
nx_dhcp_server_start	nx_dhcp_server_start(&dhcp_server); Start or resume DHCP Server processing.
nx_dhcp_server_stop	nx_dhcp_server_stop(&dhcp_server); Stop DHCP server processing.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful DHCP call.
NX_PTR_ERROR*	Invalid pointer input.
NX_DHCP_PARAMETER_ERROR	Invalid non-pointer input.
NX_DHCP_INADEQUATE_PACKET_POOL_PAYLOAD	Packet payload too small error.

NX_DHCP_NO_SERVER_OPTION_LIST	Missing option list; cannot create Server.
NX_DHCP_SERVER_BAD_INTERFACE_INDEX	Index does not match addresses.
NX_DHCP_INVALID_IP_ADDRESS	Invalid IP address or network interface for creating Server address list.
NX_DHCP_INVALID_IP_ADDRESS_LIST	Illogical start/end IP addresses for Server list.
NX_DHCP_INVALID_NETWORK_PARAMETERS	Invalid network parameters for DHCP messages to Client.
NX_DHCP_SERVER_ALREADY_STARTED	The DHCP instance has already been started.
NX_DHCP_SERVER_NOT_STARTED	DHCP Server not started.
NX_CALLER_ERROR*	Invalid caller of service.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.15.3 NetX/NetX Duo DHCP Server Module Operational Overview

The DHCP server utilizes the UDP protocol to receive DHCP Client requests and transmit responses. It handles all details of creating an IP instance, initializing the driver, creating the UDP socket, and binding to the well-known DHCP port 67 to receive client requests.

The DHCP Server is assigned a packet pool when it is created. It can share the packet pool used by the IP instance (the IP default packet pool) or the module can create a separate one for the server. The packet payload must be large enough to include DHCP data, IP and UDP headers, and the physical frame header. DHCP data size is set by the Size of the BOOT Buffer (bytes) property, which defaults to 548 bytes.

Before starting the DHCP Server, the application must create a pool of assignable IP addresses; it does so by calling the nx_dhcp_create_server_ip_address_listservice. This service takes as input a starting IP address and an ending IP address. The server verifies the addresses are local network addresses. The DHCP Server services are interface-specific, including creating the IP address list and setting network parameters. The assumed network interface the DHCP Server is running on is the primary interface (index is zero). It fills a table of IP addresses sequentially starting at the starting IP address. The addresses_added pointer input returns the number of addresses added, which is equal to or less than the size of this table. The IP address table size is defined by the Maximum size of an IP addresses list property, which defaults to 20. There is one such table for each network interface on which the DHCP Server is receiving DHCP Client requests.

The DHCP Server keeps a record of each client (or rather the client's DISCOVER request) in its client record table. The record lives for as long as the client keeps the assigned IP address. If the client fails to renew, or fails to respond to the DHCP protocol before reaching the bound (IP address assigned) state, the record is deleted. One table holds all client records from all network interfaces on which the server receives DHCP requests. The size of the table is set by the Size of client record table (units) property, which defaults to 50.

Once the DHCP Server is running and has created client records and assigned IP addresses, it periodically checks the time remaining on each of the client IP leases. The length of the IP lease is set in the Client IP address lease time (seconds) property. The default value of 0xFFFFFFFF is essentially a permanent lease. To assign leases of finite length, set the lease to a more standard time. An example lease time might be 10 days (0x0d5930 or 874,800 seconds). The interval on which the DHCP Server checks the time remaining on assigned IP leases is set to 1000 seconds*.* If a lease expires, the server simply removes the client record from the client record table, and returns that IP address back to the pool of assignable IP addresses. No message is sent to the client. The client should have initiated renew or rebind requests before its lease expired; or possibly the client has left the network.

The DHCP Server also keeps an inactivity timeout on each client session. When a client sends a packet, the inactivity timeout for that client is reset. The interval on which the DHCP Server checks the time remaining is the Fast-periodic timer interval to check valid sessions (ticks), which defaults to 10 ticks*.* This session timeout is this value multiplied by the ratio of ticks per second to produce a session timeout of 10 seconds. If a client record session time out expires, that client's IP address is returned to the pool of assignable IP addresses and the client record is cleared. No message is sent to the client.

NetX/NetX Duo DHCP Server Module Important Operational Notes and Limitations

NetX/NetX Duo DHCP Server Module Operational Notes

- The options the DHCP Server provides to the client for critical network parameters are defined in the Server option list property and set to the default value 1 3 6, which are the option codes for the Subnet Mask, Router/Gateway address, and DNS Server IP address, respectively. The number of options is set in the Server option list size property, which defaults to 3.
- The DHCP Type (Option 53) and DHCP Server Identifier (Option 54) are the DHCP parameters the server must supply to the DHCP Client.

NetX/NetX Duo DHCP Server Module Limitations

- The choice of options the DHCP Server provides are limited to some or all the following: Subnet Mask (Option 1), Router/Gateway address (Option 3), and DNS Server IP address (Option 6). Therefore, setting the Server option list size to greater than three has no effect. Setting the list of options to an option other than 1,3 or 6 has no effect.
- The NetX DHCP Server does not verify that its assignable IP addresses are not in use elsewhere in the network. It is expected that the client will check the uniqueness of its IP address it is assigned.
- The NetX DHCP Server does not support the FORCE RENEW message.
- The Relay agent field of the DHCP header is left null because the NetX DHCP Server does not support out-of-network DHCP requests.
- The DHCP Server does not correctly update the time remaining on the assigned IP leased. The slow periodic timer interval is set to 1000 ticks. Internally, that value is converted to seconds, so the actual interval on which the server checks the IP lease timeout is about 1000 * 100 assuming there are 100 ticks per second on the NetX device. If the client lease time is left at the default value of 0xFFFFFFFF, this is a permanent lease until the client decides to release it and should not be affected by this bug.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.15.4 Including the NetX/NetX Duo DHCP Server Module in an Application

This section describes how to include either or both the NetX and NetX Duo DHCP Server module in

an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo DHCP Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo DHCP Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dhcp_Server0NetXDHCPServer	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetXDHCPServer**
g_dhcp_Server0NetX Duo DHCP IPv4Server	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo DHCP IPv4Server**

When the NetX and/or NetX Duo DHCP Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

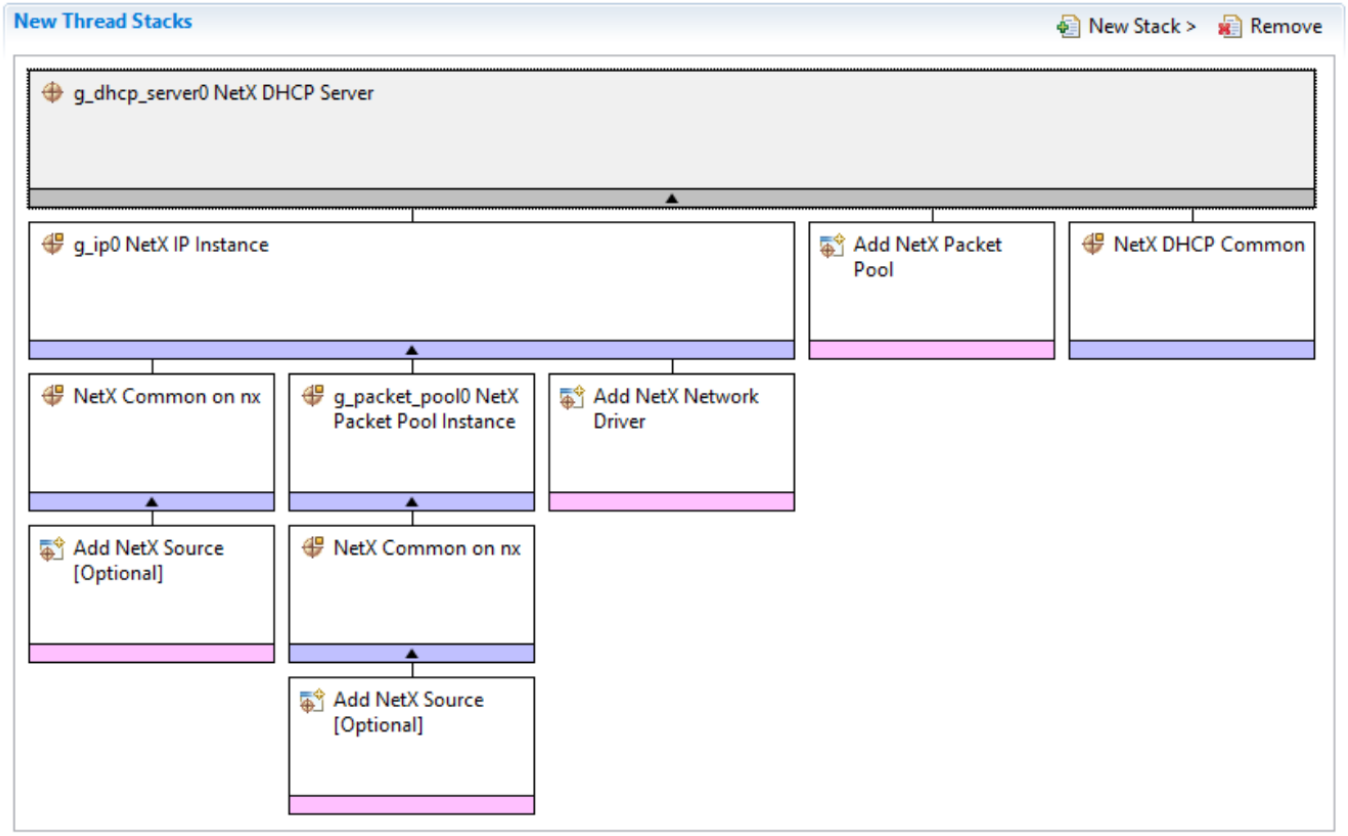


Figure 440: NetX/NetX Duo DHCP Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.15.5 Configuring the NetX/NetX Duo DHCP Server Module

The NetX/NetX Duo DHCP Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo DHCP Server Module

ISDE Property	Value	Description
Internal thread priority	1	Internal thread priority selection.
Packet allocate timeout (seconds)	2	Packet allocate timeout selection.
Fast periodic timer interval to check valid sessions (ticks)	10	Fast periodic timer interval to check valid sessions selection.
DHCP Client Session timeout - multiple of Fast periodic interval (seconds)	10	DHCP Client session timeout selection.
Client IP address default lease time (seconds)	0xFFFFFFFF	Client IP address lease time selection.
Slow periodic timer inverval to check IP lease expiration (seconds)	1000	Slow periodic timer inverval to check IP lease expiration selection.
Size of the array to contain options in client request (units)	12	Size of the array containing current requested options selection.
Server option list (optional - use space for separation	1 3 6	Module server option list selection.
Server option list size (optional)	3	Server option list size selection.
Size of the server host main buffer (bytes)	32	Size of the server host main buffer selection.
Size of the current client hostname buffer (byte)	32	Size of the current client hostname buffer selection.
Maximum size of an IP addresses list (units)	20	Maximum size of an IP addresses list selection.
Size of the client record table (units)	50	Size of the client record table selection.
Size of the BOOT buffer (bytes)	548	Size of the BOOT buffer selection.
Name	g_dhcp_server0	Module name.
Internal thread stack size (bytes)	NetX Default: 2048 NetX Duo Default: 4096	Internal thread stack size selection.
Name of generated initialization function	nx_dhcp_server_init0	Name of generated initialization function selection.

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different Ethernet interface pins and resets. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo DHCP Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.

TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo DHCP Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragmentation option selection.
Time to live	128	Time to live selection.
Packet Queue depth	5	Packet queue depth selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo DHCP Server Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo DHCP Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence

ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.15.6 Using the NetX/NetX Duo DHCP Server Module in an Application

The steps in using the NetX/NetX Duo DHCP Server module in a typical application are:

1. Create a pool of assignable IP addresses using the nx_dhcp_create_server_ip_address_list API.
2. Set network parameters that will be returned by the server using the nx_dhcp_set_interface_network_parameters API.
3. Start the DHCPv4 server with the nx_dhcp_server_start API.

The following figure illustrates common steps in a typical operational flow diagram:

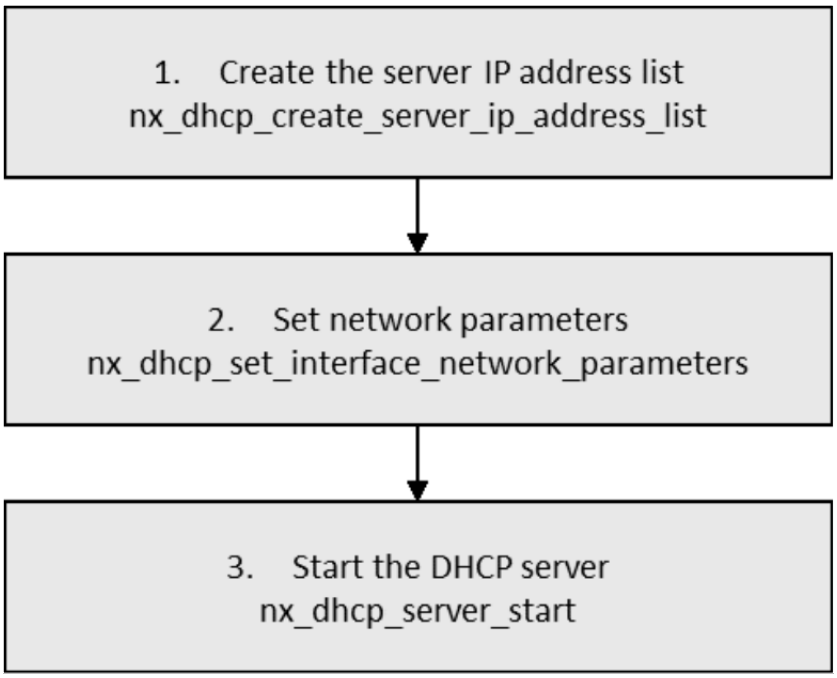


Figure 441: Flow Diagram of a Typical NetX/NetX Duo DHCP Server Module Application

4.3.16 NetX Duo DHCPv6 Client

4.3.16.1 NetX Duo DHCP IPv6 Client Introduction

The Dynamic Host Configuration Protocol (DHCP) is used to obtain an IP address and network parameters. The DHCP is designed to extend the basic functionality of the BOOTP (which is limited to static address configuration) to include a completely dynamic IP address allocation through "leasing" an IP address to a client for a specified period of time. The DHCP can also be configured to allocate IP addresses in a static manner (like the BOOTP). An application's IP address is one of the supplied parameters for the NetX™ component. Supplying the IP address poses no problem if the IP address is known to the application, either statically or through the user configuration. When the application does not know or care what its IP address is, the NetX is initialized with a zero IP address; a DHCP client component added to NetX can then dynamically obtain an IP address.

The document covers the NetX Duo DHCPv6 Client API and how it is used to obtain IPv6 addresses. In IPv6 networks, DHCPv6 (instead of DHCP) is used for dynamic global IPv6 address assignment from a DHCPv6 server. DHCPv6 offers many of the same features, as well as several enhancements.

NetX Duo DHCP IPv6 Client Module Features

- NetX Duo DHCPv6 Client is compliant with RFC 3315, RFC 3646, and related RFCs.
- Provides high-level APIs for:
    - Creating and deleting a DHCPv6 Client instance
    - Starting and stopping a DHCPv6 Client
    - Message sending and processing
    - Retrieving DHCPv6 data from the DHCPv6 Client

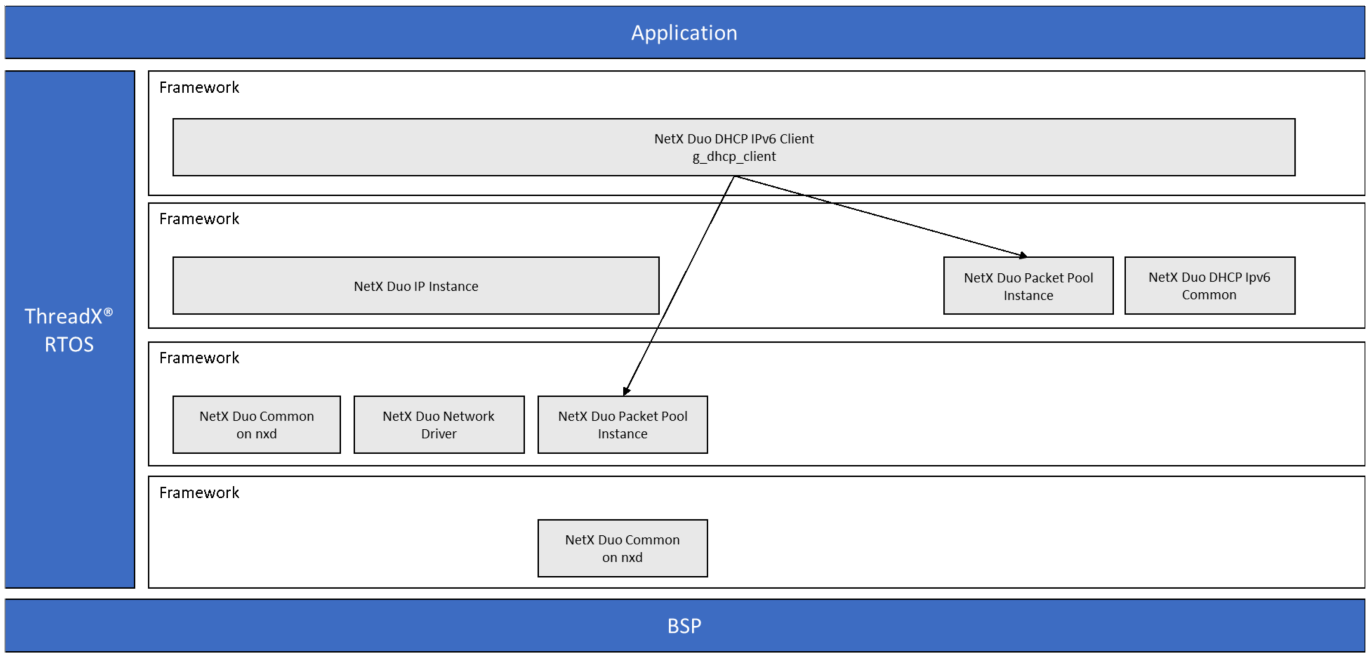


Figure 442: NetX Duo DHCP IPv6 Client Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo DHCP IPv6 Client Module in an Application for additional details.

4.3.16.2 NetX Duo DHCP IPv6 Client Module APIs Overview

The NetX Duo DHCPv6 Client framework defines APIs for creating, deleting, adding and getting client information. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo DHCP IPv6 Client Module API Summary

Function Name	Example API Call and Description
nx_dhcpv6_client_create	nx_dhcpv6_client_create(&dhcp_0, &ip_0, "DHCPv6 Client", &pool_0,NULL, NULL, pointer, 2048, dhcpv6_state_change_notify, dhcpv6_server_error_handler); Create a DHCPv6 Client instance.
nx_dhcpv6_client_delete	nx_dhcpv6_client_delete(&my_dhcp); Delete a DHCPv6 Client instance.
nx_dhcpv6_client_set_interface	nx_dhcpv6_client_set_interface(&dhcp_0, index); Set the Client network interface for communications with the DHCPv6 Server.
nx_dhcpv6_create_client_duid	nx_dhcpv6_create_client_duid(&dhcp_0, NX_DHCPV6_DUID_TYPE_LINK_TIME, NX_DHCPV6_HW_TYPE_IEEE_802, 0) Create a DHCPv6 Client DUID.

nx_dhcpv6_create_client_ia	nx_dhcpv6_create_client_ia(&dhcp_0, &ipv6_address, NX_DHCPV6_PREFERRED_LIFETIME, NX_DHCPV6_VALID_LIFETIME); Legacy Add a DHCPv6 Client Identity Address (IA).
nx_dhcpv6_create_client_iana	nx_dhcpv6_create_client_iana(&dhcp_0, DHCPV6_IA_ID, DHCPV6_T1, DHCPV6_T2); Create a DHCPv6 Client Identity Association for Non-Temporary Addresses (IANA).
nx_dhcpv6_add_client_ia	nx_dhcpv6_add_client_ia(&dhcp_0, &ipv6_address, NX_DHCPV6_PREFERRED_LIFETIME, NX_DHCPV6_VALID_LIFETIME); Add a DHCPv6 Client Identity Address (IA).
nx_dhcpv6_get_client_duid_time_id	nx_dhcpv6_get_client_duid_time_id(&dhcp_0, &time_ID); Get the time ID from DHCPv6 Client DUID.

nx_dhcpv6_get_ip_address	nx_dhcpv6_get_IP_address(&dhcp_0, &ipv6_address);
	nxd_ipv6_address_set(&ip_0, 0, &ipv6_address, 64, &address_index);
	Get the global IPv6 address assigned to the DHCPv6 client.

nx_dhcpv6_get_lease_time_data	nx_dhcpv6_get_lease_time_data(&dhcp_0, &T1, &T2, &preferred_lifetime, &valid_lifetime); Get T1 and T2 in the Identity Association (IANA) leased to the DHCPv6 Client.
nx_dhcpv6_get_iana_lease_time	nx_dhcpv6_get_iana_lease_time(&dhcp_0, &T1, &T2); Get T1, T2, valid and preferred lifetimes for the DHCPv6 Client IPv6 address by address index.
nx_dhcpv6_get_valid_ip_address_count	nx_dhcpv6_get_valid_ip_address_count(&dhcp_0, &address_count); This service retrieves the count of the Client's valid IPv6 addresses. A valid IPv6 address is bound (assigned) to the Client and registered with the IP instance. Also useful for determining if the DHPCv6 Client has reached the bound state.
nx_dhcpv6_get_valid_ip_address_lease_time	nx_dhcpv6_get_valid_ip_address_lease_time(&dhcp_0, &ip_address, &preferred_lifetime, &valid_lifetime); Get T1, T2, valid and preferred lifetimes for the DHCPv6 Client IPv6 address by address index.
nx_dhcpv6_get_DNS_server_address	nx_dhcpv6_get_DNS_server_address(&dhcp_0, index, &server_address); Get DNS Server address at the specified index into the DHCPv6 Client DNS server list.

nx_dhcpv6_get_other_option_data	nx_dhcpv6_get_other_option_data(&dhcp_0, option_code, buffer); Get the specified option data, such as domain name or time zone server.
nx_dhcpv6_get_time_accrued	nx_dhcpv6_get_time_accrued(&dhcp_0, &time_accrued); Get the time accrued the global IPv6 address lease has been bound to the DHCPv6 Client.
nx_dhcpv6_get_time_server_address	nx_dhcpv6_get_time_server_address(&dhcp_0, index, &server_address); Get Time Server address at the specified index into the DHCPv6 Client Time server list.
nx_dhcpv6_reinitialize	nx_dhcpv6_reinitialize(&dhcp_0); Reinitialize the DHCPv6 for restarting the DHCPv6 Client state machine and rerunning the DHCPv6 protocol.
nx_dhcpv6_request_confirm	nx_dhcpv6_request_confirm(&dhcp_0); Send a CONFIRM request to the Server.
nx_dhcpv6_request_inform_request	nx_dhcpv6_request_inform_request(&dhcp_0); Send an INFORM REQUEST message to the Server.
nx_dhcpv6_request_option_DNS_server	nx_dhcpv6_request_option_DNS_server(&dhcp_0, NX_TRUE); Add the DNS server option to the Client option request data in request messages to the Server.
nx_dhcpv6_request_option_FQDN	nx_dhcpv6_request_option_FQDN(&dhcp_0, "DHCPv6_Client", NX_DHCPV6_CLIENT_DESIRES_NO_SERVER_DNS_UPDATE); Add the FQDN option to the Client option request data in request messages to the Server.
nx_dhcpv6_request_option_domain_name	nx_dhcpv6_request_option_domain_name(&dhcp_0, NX_TRUE); Add the domain name option to the Client option request data in request messages to the Server.
nx_dhcpv6_request_option_time_server	nx_dhcpv6_request_option_time_server(&dhcp_0, NX_TRUE); Add the time server option to the Client option request data in request messages to the Server.
nx_dhcpv6_request_option_timezone	nx_dhcpv6_request_option_timezone(&dhcp_0, NX_TRUE); Add the time zone option to the Client option request data in request messages to the Server.
nx_dhcpv6_request_release	nx_dhcpv6_request_release(&dhcp_0); Send a RELEASE request to the Server.
nx_dhcpv6_request_solicit	nx_dhcpv6_request_solicit(&dhcp_0); Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast).

nx_dhcpv6_request_solicit_rapid	nx_dhcpv6_request_solicit_rapid(&dhcp_0); Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast) with the Rapid Commit option set.
nx_dhcpv6_resume	nx_dhcpv6_resume(&dhcp_0); Resume DHCPv6 Client processing.
nx_dhcpv6_set_time_accrued	nx_dhcpv6_set_time_accrued(&dhcp_0, time_accrued); Set the time accrued on the global Client IPv6 address lease in the Client record.
nx_dhcpv6_start	nx_dhcpv6_start(&dhcp_0); Start the DHCPv6 Client thread task. Note this is not equivalent to starting the DHCPv6 state machine and does not send a SOLICIT request.
nx_dhcpv6_stop	nx_dhcpv6_stop(&dhcp_0); Stop the DHCPv6 Client thread task.
nx_dhcpv6_suspend	nx_dhcpv6_suspend(&dhcp_0); Suspend the DHCPv6 Client thread task.
The following services are available if NX_DHCPV6_CLIENT_RESTORE_STATE is defined for the project:	
nx_dhcpv6_client_get_record	nx_dhcpv6_client_get_record(dhcpv6_ptr, client_record_ptr); Obtain a record of the client state (to save to non-volatile memory)
nx_dhcpv6_client_restore_record	nx_dhcpv6_client_restore_record(dhcpv6_ptr, client_record_ptr, time_elapsed); Apply saved client record to the current Client instance. Note the DHCPv6 Client thread must be not be running when this task is called.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful API call.
NX_PTR_ERROR*	Invalid pointer input.
NX_CALLER_ERROR*	Must be called from thread.
NX_DHCPV6_PARAM_ERROR	Invalid non pointer input.
NX_INVALID_INTERFACE	Invalid interface index input.
NX_DHCPV6_UNSUPPORTED_DUID_TYPE	DUID type unknown or not supported.

NX_DHCPV6_UNSUPPORTED_DUID_HW_TYPE	DUID hardware type unknown or not supported.
NX_DHCPV6_IA_ADDRESS_ALREADY_EXIST	Duplicate IA address.
NX_DHCPV6_REACHED_MAX_IA_ADDRESS	IA exceeds the max IAs Client can store.
NX_DHCPV6_INVALID_IA_ADDRESS	Invalid (for example, null) IA address in IA.
NX_DHCPV6_IA_ADDRESS_NOT_VALID	IPv6 address successfully assigned.
NX_DHCPV6_UNKNOWN_OPTION	Unknown/unsupported option code.
NX_DHCPV6_ALREADY_STARTED	DHCPv6 Client is already running.
NX_DHCPV6_NOT_STARTED	DHCPv6 Client task not started.
NX_DHCPV6_MISSING_REQUIRED_OPTIONS	Client missing required options.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.16.3 NetX Duo DHCP IPv6 Client Module Operational Overview

The DHCPv6 protocol also uses the UDP to dynamically obtain IPv6 addresses. A NetX Duo IP instance is created automatically, and UDP, IPv6, and ICMPv6 are enabled on the IP instance prior to creating the DHCPv6 Client. When the DHCPv6 Client is created, a UDP socket is created and bound to port 546. The *IPv6 Global Address* property of the IP instance is locked to a zero IPv6 address; otherwise, the IPv4 address may be any network IP address. If the IP instance *IPv6 Link Local Address* property is set to zero, the DHCPv6 Client creates the link local address from the MAC address. The MAC address is set in the NetX Port ETHER instance. See *Channel 1 MAC Address High Bits* and *Channel 1 MAC Address Low Bits* properties. This is the source address for DHCPv6 messages to the server.

To begin the process of requesting a global IPv6 address assignment, a Client first broadcasts a SOLICIT message using the nx_dhcpv6_request_solicit service. In IPv6, the broadcast address is the All_DHCP_Relay_Agents_and_Servers address (FF02::1:2.). A DHCPv6 Server responds with an ADVERTISE message containing a global IPv6 address (not a link local address) for the Client, the IPv6 address lease time, and any additional information requested by the client. The DHCPv6 protocol requires the client to wait for a period of time to receive ADVERTISE messages from all DHCPv6 Servers on the network. The client pre-processes each ADVERTISE message to be a valid message and scans the option data for various DHCPv6 parameters; it also checks the preference value in the preference option, if supplied by the server. If more than one ADVERTISE message is received, the NetX DHCPv6 Client chooses the ADVERTISE message with the highest preference value received by the end of the wait period. If the Client receives an ADVERTISE message with a preference value of 255, it accepts that message immediately and discards all subsequent ADVERTISE messages.

The client extracts data from the ADVERTISE message and broadcasts (so all DHCPv6 Servers are informed) a REQUEST message specifying which server the client chooses; that server then confirms the assigned address information and lease times with a REPLY message to complete the protocol.

The DHCPv6 Client is promoted to the bound state and automatically registers the assigned IPv6 address with the IP instance.

Notification of Successful Address Assignment and Validation

The application can determine when the DHCPv6 Client is bound to an IPv6 address in two ways: the first is to query the DHCPv6 Client itself for a valid IPv6 address using the nx_dhcpv6_get_IP_address service for clients for applications using only one global IPv6 address assigned, (the general case) or nx_dhcpv6_get_valid_ip_address_count service for clients with more than one IPv6 address assigned. The second way requires the DHCPv6 Client be configured with the state change callback, the *Name of state change notification function* property* of the DHCPv6 Client stack element.

If the DHCPv6 Client receives a response from the server, but the server is unable to assign the address, the server returns an error status. The application is notified of the error status received if configured with the DHCPv6 Client server-error callback - the *Name of server error handler* property of the DHCPv6 Client stack element.

These callbacks must be defined by the application because these callback functions are called from the DHCPv6 Client thread task; the client application must NOT call any NetX Duo DHCPv6 Client services that require mutex control of the DHCPv6 Client (such as nx_dhcpv6_start, nx_dhcpv6_stop), and any of the APIs that send messages directly from the callback (such as nx_dhcpv6_request_release).

The states of the DHCP IPv6 protocol are:

NX_DHCPV6_STATE_INIT

NX_DHCPV6_STATE_SENDING_SOLICIT

NX_DHCPV6_STATE_SENDING_REQUEST

NX_DHCPV6_STATE_SENDING_RENEW

NX_DHCPV6_STATE_SENDING_REBIND

NX_DHCPV6_STATE_SENDING_DECLINE

NX_DHCPV6_STATE_SENDING_INFORM_REQUEST

NX_DHCPV6_STATE_BOUND_TO_ADDRESS

Duplicate Address Detection of the IPv6 address

If configured for the Duplicate Address Detection (DAD) protocol, enabled by default in NetX Duo *Duplicate Address Detection property*, NetX Duo automatically sends "Neighbor Solicit" messages to verify the assigned address is unique on the network. If the IPv6 address is unique, NetX Duo notifies the DHCPv6 Client when the assigned address has been promoted from NX_IPV6_ADDR_STATE_TENTATIVE to NX_IPV6_ADDR_STATE_VALID internally. The application must allow time for the DAD to finish processing, which takes about 4-5 seconds. If the DAD is not enabled, the IP instance marks the address VALID immediately.

Once an IPv6 address is valid, the device may use that IPv6 address to send and transmit IPv6 messages.

If the DAD protocol fails, NetX Duo notifies the DHCPv6 Client to send a DECLINE message to the

server and restart the DHCPv6 Client at the INIT state.

Retransmission of DHCPv6 Client Solicitations

The DHCPv6 Client times out waiting for a server reply before sending another DHCPv6 message and defaults to 1 second on the first retransmit, per RFC 3315 recommendations. If the DHCPv6 Client fails to receive a valid server response to the SOLICIT message, each subsequent retransmission interval doubles up to a maximum of 120 seconds by default.

DHCPv6 Lease Timeouts

The IPv6 lease assigned by the server contains two timeout parameters (T1 and T2) in the DHCPv6 Client Identity Association – Non-Temporary Addresses, (IANA), which is the data type specified by RFC 3315 to store IPv6 address data in DHCPv6.

When the time elapsed on an assigned IPv6 address reaches T1, the DHCPv6 Client automatically sends a RENEW message. If the elapsed time reaches T2 without a successful renewal, DHCPv6 Client automatically sends a REBIND message. If it still receives no response, the DHCP Client unregisters the IPv6 address with the IP instance and restarts the DHCPv6 protocol at the INIT state. Two other IPv6 lease parameters, preferred and valid lifetime, are assigned automatically to the Identity Association (IA) contained in the IANA in the DHCPv6 process. When the preferred and valid lifetimes expire, the assigned IPv6 address is either deprecated or rendered invalid; meaning a valid T1 must be less than the preferred lifetime and a T2 must be less than the valid lifetime.

NetX Duo DHCP IPv6 Client Module Important Operational Notes and Limitations

NetX Duo DHCP IPv6 Client Module Operational Notes

- The NetX Duo Source element has a few key properties for supporting DHCPv6: The *NetX Duo IPv6 Support* property, the *Checksum computation support on received ICMPV6 packets,* and the *Checksum computation support on transmitted ICMPv6 packets*. The latter two ensure that incoming and outgoing packets have an ICMPv6 checksum in the ICMPv6 header; these are automatically enabled for the DHCPv6 Client module. If the NetX Duo Source element is added to the project, check to make sure these properties are enabled.
- The DHCPv6 Client creation requires a previously created packet pool. The application can use the IP default packet pool (g_packet_pool0) used by the IP instance or it can create its own (usually g_packet_pool1.)
- Before sending a SOLICIT request, the Client must create a DHCP Unique Identifier (DUID) to uniquely define the client on the network. The MAC address is usually used but can be another unique identifier. A typical invocation of this service is:

```
nx_dhcpv6_create_client_duid(&g_dhcpv6_client0,
NX_DHCPV6_DUID_TYPE_LINK_TIME /* Use MAC address */,
              NX_DHCPV6_HW_TYPE_IEEE_802,
              0 /* Client DUID time, usually set to zero */);
```

- The DHCPv6 Client must also create the IANA for the client; this structure holds IPv6 lease information such as IPv6 addresses and T1 and T2 times. (The client can use the IANA to request lease times.) To create an IANA, use thenx_dhcpv6_client_create_iana service:

```
status = nx_dhcpv6_create_client_iana(&g_dhcpv6_client0,
DHCPV6_IANA_ID /* ULONG unique ID */,
DHCPV6_T1 /* Request T1 time in seconds or
             set to TX__WAIT_FOREVER */,
DHCPV6_T2 /* Request T2 time;must be longer
          than T1 */);
```

- In the SOLICIT request, the client may request the assignment of a specific IPv6 address from the server by calling the nx_dhcpv6_add_client_ia service before calling nx_dhcpv6_request_solicit. This service uses an NXD_ADDRESS address data type for the IPv6 address. See *NetX Duo User Guide for the Renesas Synergy™ Platform* for details on the data type definition in NetX Duo.
- To request network information such as a DNS server, NTP server, and other options, the application can all these APIs before calling nx_dhcpv6_request_solicit:
    - nx_dhcpv6_request_option_timezone(&g_dhcpv6_client, NX_TRUE);
    - nx_dhcpv6_request_option_DNS_server(&g_dhcpv6_client, NX_TRUE);
    - nx_dhcpv6_request_option_time_server(&g_dhcpv6_client, NX_TRUE);
    - nx_dhcpv6_request_option_domain_name(&g_dhcpv6_client, NX_TRUE);
- If the client needs to release an assigned IPv6 address, it informs the DHCPv6 server by calling the nx_dhcpv6_request_release service. The DHCPv6 Client sends a unicast RELEASE message to the server and should wait for the server REPLY.
- For DHCPv6 Client services that retrieve information about the DHCPv6 Client, an address index may need to be specified. Most clients have one IPv6 global address assigned, so the address index is 0.
- To obtain specific information about lease times, use the nx_dhcpv6_get_valid_ip_address_lease_time service. This requires an address index input (usually 0.)

NetX Duo DHCP IPv6 Client Module Limitations

- The NetX Duo DHCPv6 Client does not support the server unicast option for sending unicast DHCPv6 messages to the DHCPv6 Server even if the server indicates this is permitted.
- The NetX Duo DHCPv6 Client only supports DUIDs for LINK (MAC address) and LINK TIME (MAC address and time input.)
- The NetX Duo DHCPv6 Client does not support the reconfigure request in which a server initiates IPv6 address changes to the clients on the network.
- The NetX Duo DHCPv6 Client does not support the enterprise format for the DHCPv6 unique identifier control block; it only supports Link Layer and Link Layer Plus Time formats.
- The NetX Duo DHCPv6 Client does not support Temporary Association (TA) address requests, but does support Non Temporary (IANA) option requests.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.16.4 Including the NetX Duo DHCP IPv6 Client Module in an Application

This section describes how to include either or both the NetX Duo DHCP IPv6 Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo DHCP IPv6 Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo DHCP IPv6 Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dhcp_client0 NetX Duo DHCP IPv6 Client	Threads	**New Stack> X-Ware> NetX Duo> Protocols> NetX Duo DHPC IPv6 Client**

When the NetX Duo DHCP IPv6 Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

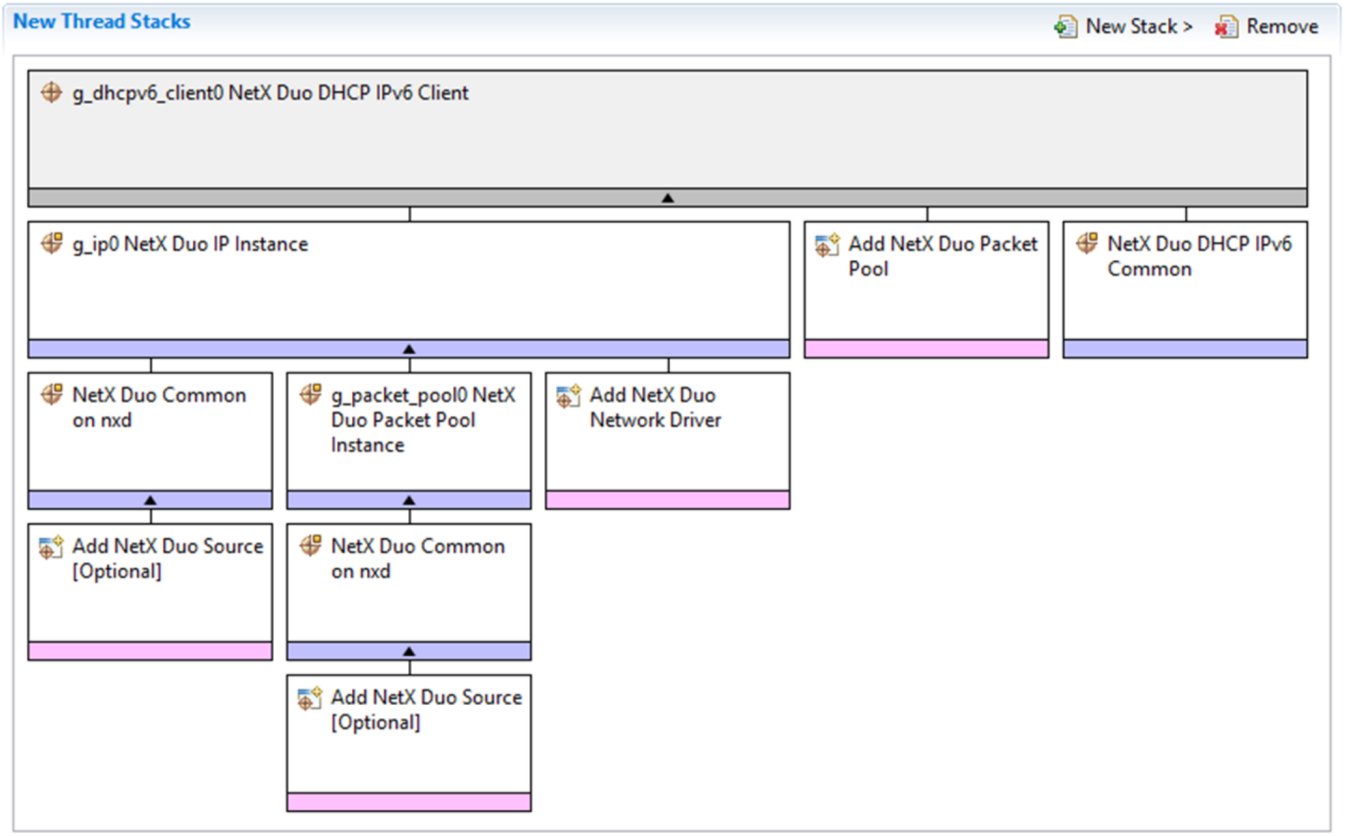


Figure 443: NetX Duo DHCP IPv6 Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.16.5 Configuring the NetX Duo DHCP IPv6 Client Module

The NetX Duo DHCP IPv6 Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo DHCP IPv6 Client Module

ISDE Property	Value	Description
Internal Thread Priority	3	Internal thread priority selection.
Time out for obtaining DHCPv6 client mutex (ticks)	TX_WAIT_FOREVER	Time out for obtaining DHCPv6 client mutex selection.
Time interval between current IP address lease time update (seconds)	1	Time interval between current IP address lease time update selection.
Maximum IA addresses allowed in client record	1	Maximum IA addresses allowed in client record selection.
Number of DNS servers the client will store	2	Number of DNS servers the client will stored selection.
Number of time servers the client will store	1	Number of time servers the client will store selection.
Domain name buffer size (bytes)	32	Domain name buffer size selection.
Current time zone information buffer size (bytes)	16	Current time zone information buffer size selection.

Maximum DHCPv6 server messages buffer size (bytes)	100	Maximum DHCPv6 server messages buffer size selection.
Name	g_dhcpv6_client0	Module name.
Internal thread stack size (bytes)	4096	Internal thread stack size selection.
Name of state change notification function	dhcpv6_state_change_notify	Name of state change notification function selection.
Name of server error handler	dhcpv6_server_error_handler	Name of server error handler selection.
Name of generated initialization function	nx_dhcpv6_client_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different Ethernet interface pins and resets. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo DHCP IPv6 Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.

IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo DHCP IPv6 Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliabililty, Minimum cost Default: Normal	Type of service UDP requests selection

Time to live	128	Time to live selection
Packet Queue depth	5	Packet queue depth selection
packet alocation timeout (seconds)	3	Packet allocation timeout selection
Interval for active session time update (seconds)	3	Interval for active session time update selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo DHCP IPv6 Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo DHCP IPv6 Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.16.6 Using the NetX Duo DHCP IPv6 Client Module in an Application

The following example assumes a system that is already established with a working and enabled IP,

ARP and UDP, and the link is running.

The steps in using the NetX Duo DHCP IPv6 Client module in a typical application are:

1. Create a Client DUID for the DHCPv6 Client using the nx_dhcpv6_create_client_duid.
2. Create an IANA for the DHCPv6 Client using the nx_dhcpv6_create_client_iana.
3. Request network options (such as DNS server and time server) using the nx_dhcpv6_request_option_DNS_server and nx_dhcpv6_request_option_time_server API [Optional].
4. Start the DHCPv6 Client with the nx_dhcpv6_start API. This sets the DHCPv6 Client state and binds the client socket, in effect getting the DHCPv6 Client ready to exchange messages with the server.
5. Send the SOLICIT message to the server using the nx_dhcpv6_request_solicit API. The DHCPv6 Client internally manages the rest of the DHCPv6 protocol.
6. Check for IPv6 Address resolution using the nx_dhcpv6_get_valid_ip_address_count API to return with an address_count > 0. The DHCPv6 Client now as a valid IPv6 address.

The following figure illustrates common steps in a typical operational flow diagram:

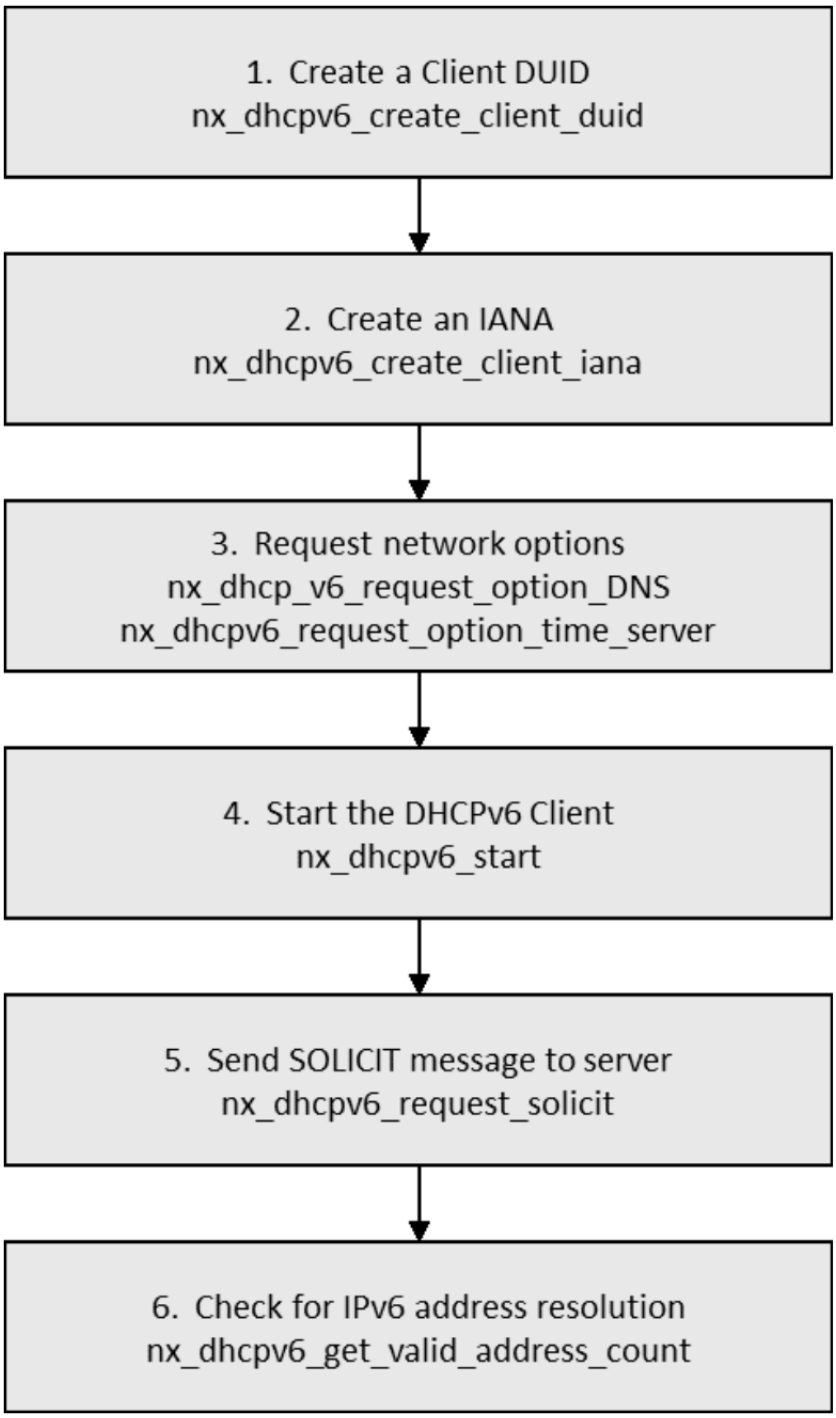


Figure 444: Flow Diagram of a Typical NetX Duo DHCP IPv6 Client Module Application

4.3.17 NetX Duo DHCPv6 Server

4.3.17.1 NetX Duo DHCP IPv6 Server Introduction

The Dynamic Host Configuration Protocol (DHCP) is used to obtain an IP address and network parameters. The DHCP is designed to extend the basic functionality of the BOOTP (which is limited to static address configuration) to include a completely dynamic IP address allocation through "leasing" an IP address to a client for a specified period of time. The DHCP can also be configured to allocate IP addresses in a static manner (like the BOOTP). An application's IP address is one of the supplied parameters for the NetX™ component. Supplying the IP address poses no problem if the IP address is known to the application, either statically or through the user configuration. When the application does not know or care what its IP address is, the NetX is initialized with a zero IP address; a DHCP client component added to NetX can then dynamically obtain an IP address.

In IPv6 networks, DHCPv6 replaces DHCP (which is limited to IPv4) for dynamic global IPv6 address assignment from a DHCPv6 Server. The DHCPv6 offers most of the same features, as well as many enhancements, and explains how the NetX Duo™ DHCPv6 Server API is used for DHCPv6 Client IPv6 address requests.

NetX Duo DHCP IPv6 Server Module Features

- The NetX Duo DHCPv6 Server is compliant with RFC 3315, RFC 3646 and related RFCs.
- Provides high-level APIs for:
    - Creating and deleting a DHCPv6 Server instance
    - Starting and stopping a DHCPv6 Server thread task
    - Creating a pool of IPv6 addresses for lease
    - Maintaining a table of DHCPv6 leases assignable to requesting clients

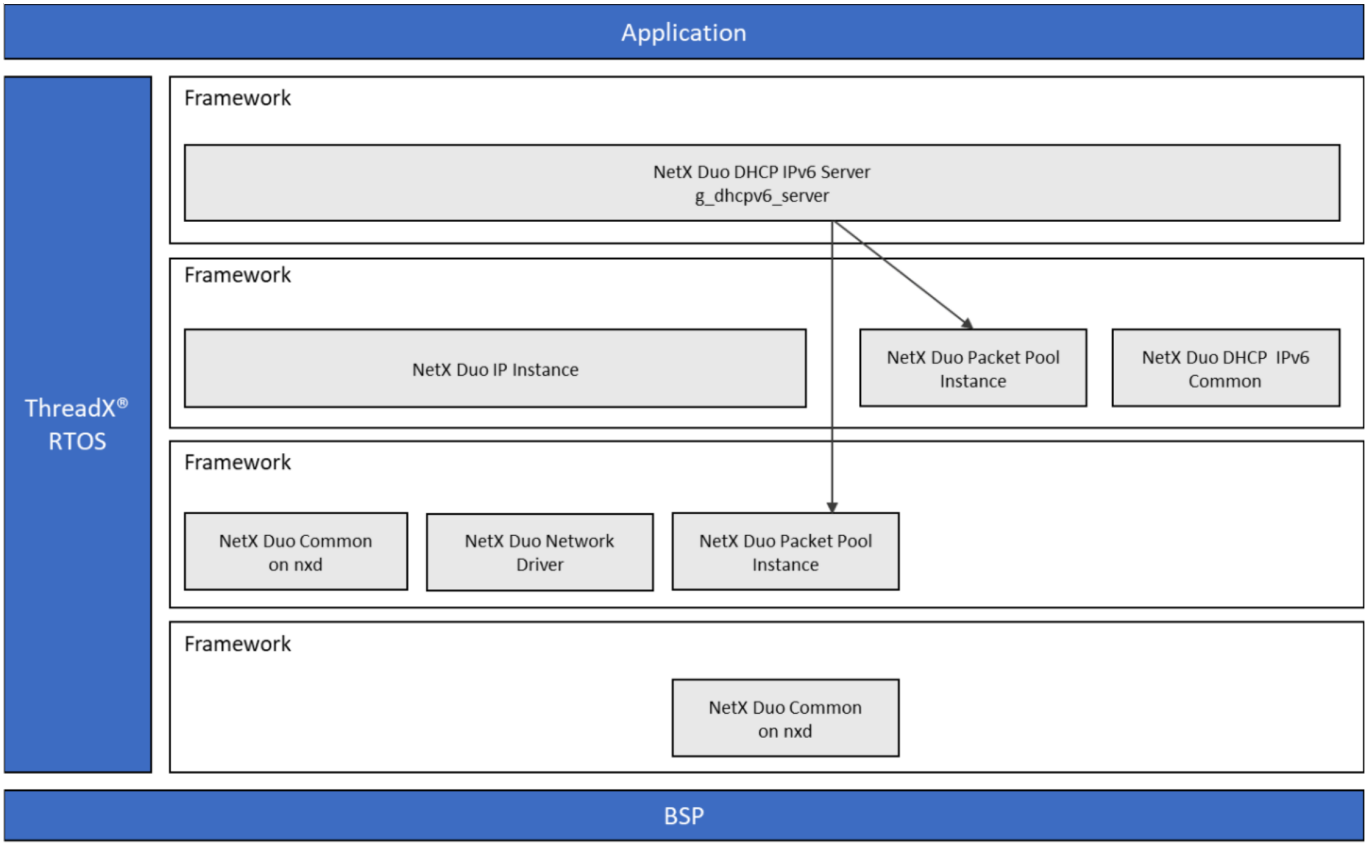


Figure 445: NetX Duo DHCP IPv6 Server Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo DHCP IPv6 Server Module in an Application for additional details.

4.3.17.2 NetX Duo DHCP IPv6 Server Module APIs Overview

The NetX Duo DHCPv6 Server module defines APIs for creating, deleting, adding and getting server information. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo DHCP IPv6 Server Module API Summary

Function Name	Example API Call and Description
nx_dhcpv6_server_create	nx_dhcpv6_server_create(&g_dhcpv6_server0, &server_ip_address, "DHCPv6 Server", &g_packet_pool0, stack_pointer, NX_DHCPV6_SERVER_THREAD_STACK_SIZE, address_declined_handler, option_request_handler); Create a DHCPv6 Server instance.
nx_dhcpv6_server_delete	nx_dhcpv6_client_delete(&g_dhcpv6_server0); Delete a DHCPv6 Server instance and release resources (unbind port, delete socket, timers and thread).
nx_dhcpv6_create_ip_address_lease	nx_dhcpv6_create_ip_address_range(&g_dhcpv6_server0, &start_ipv6_address, &end_ipv6_address, &addresses_added); Create the Server's IPv6 address lease pool.
nx_dhcpv6_reserve_ip_address_range	nx_dhcpv6_reserve_ip_address_range(&g_dhcpv6_server0, &start_ipv6_address, &end_ipv6_address, &addresses_reserved); Reserve the specified range of IPv6 addresses not to be leased out to a requesting Client.
nx_dhcpv6_add_ip_address_lease	nx_dhcpv6_add_ip_address_lease(&g_dhcpv6_server0, table_index, &lease_IP_address, T1, T2, valid_lifetime, preferred_lifetime); Copy an IPv6 lease record into the specified index into the Server table. Intended for use in non-volatile storage of IPv6 lease data.
nx_dhcpv6_add_client_record	nx_dhcpv6_add_client_record(&g_dhcpv6_server0, table_index, message_xid, &client_address, client_state, IP_lease_time_accrued, valid_lifetime, duid_type, duid_hardware, physical_address_msw, physical_address_lsw, duid_time, duid_vendor_number, duid_vendor_private, duid_private_length); Copy a Client record into the specified index into the Server table. Intended for use in non-volatile storage of Client IPv6 address data.

nx_dhcpv6_create_dns_address	nx_dhcpv6_create_dns_address(&g_dhcpv6_server0, &dns_ipv6_address); Set the DNS server address to include in network parameters sent to clients.
nx_dhcpv6_retrieve_client_record	nx_dhcpv6_retrieve_client_record(&g_dhcpv6_server0, table_index, message_xid, &client_address, client_state, IP_lease_time_accrued, valid_lifetime, duid_type, duid_hardware, physical_address_msw, physical_address_lsw, duid_time, duid_vendor_number, duid_vendor_private, duid_private_length); Retrieve items from the Client specified by the index into the Server table. Intended for use in non-volatile storage of Client IPv6 address data.
nx_dhcpv6_retrieve_ip_address_lease	nx_dhcpv6_retrieve_ip_address_lease(&g_dhcpv6_server0, table_index, &lease_IP_address, T1, T2, valid_lifetime, preferred_lifetime); Retrieve items from the IPv6 lease specified by the index into the Server table. Intended for use in non-volatile storage of IPv6 lease data.
nx_dhcpv6_server_interface_set	nx_dhcpv6_server_interface_set(&g_dhcpv6_server0, 0, 1); Set the interface the DHCPv6 Server will run on, and the global address the DHCPv6 Server will use in messages to Clients. By default, the DHCPv6 Server runs on the primary interface (index 0).
nx_dhcpv6_set_server_duid	nx_dhcpv6_set_server_duid(&g_dhcpv6_server0, NX_DHCPV6_SERVER_DUID_TYPE, NX_DHCPV6_SERVER_HW_TYPE, physical_address_msw, physical_address_lsw, duid_time); Create the Server DUID which is a required part of the DHCPv6 header and uniquely identifies the DHCPv6 Server.
nx_dhcpv6_server_start	nx_dhcpv6_server_start(&g_dhcpv6_server0); Start the DHCPv6 thread task.
nx_dhcpv6_server_suspend	nx_dhcpv6_server_suspend(&g_dhcpv6_server0); Suspend the DHCPv6 server task.
nx_dhcpv6_server_resume	nx_dhcpv6_server_resume(&g_dhcpv6_server0); Resume the DHCPv6 server task.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful API call.
NX_PTR_ERROR*	Invalid pointer input.
NX_CALLER_ERROR*	Must be called from thread.
NX_DHCPV6_PARAM_ERROR	Invalid non pointer input.
NX_INVALID_INTERFACE	Invalid interface index input.
NX_DHCPV6_UNSUPPORTED_DUID_TYPE	DUID type unknown or not supported.
NX_DHCPV6_UNSUPPORTED_DUID_HW_TYPE	DUID hardware type unknown or not supported.
NX_DHCPV6_IA_ADDRESS_ALREADY_EXIST	Duplicate IA address.
NX_DHCPV6_REACHED_MAX_IA_ADDRESS	IA exceeds the max IAs Client can store.
NX_DHCPV6_INVALID_IA_ADDRESS	Invalid (for example, null) IA address in IA.
NX_DHCPV6_IA_ADDRESS_NOT_VALID	IPv6 address successfully assigned.
NX_DHCPV6_UNKNOWN_OPTION	Unknown/unsupported option code.
NX_DHCPV6_ALREADY_STARTED	DHCPv6 Client is already running.
NX_DHCPV6_NOT_STARTED	DHCPv6 Client task not started.
NX_DHCPV6_MISSING_REQUIRED_OPTIONS	Client missing required options.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.17.3 NetX Duo DHCP IPv6 Server Module Operational Overview

The NetX Duo DHCPv6 Server module creates a NetX Duo IP instance for the server and a UDP socket bound to the well-known DHCPv6 Server port 547 to listen for client requests. Before starting DHCPv6, the server needs a global IPv6 address by setting the IPv6 Global Address property in the IP NetX Duo Instance property box. (Note that this is a 128-bit long address, compared to the 32-bit long IPv4 address.)

The DHCPv6 Server should wait for the IPv4 address to be validated using the nx_ip_status_check service, even if the server does not use this IP address for DHCPv6 messages. The driver needs to be initialized with information from the IP layer and the link needs to be enabled, all of which happens with IPv4 address-registration.

Before an application can start the DHCPv6 Server, it must create a pool of assignable IPv6 addresses using the nx_dhcpv6_create_ip_address_range service. The application must also create a server DUID (DHCP Unique Identifier, usually based on the MAC address, and is required in all DHCPv6 Server messages using the nx_dhcp_set_server_duid service. Optionally, it can set the network DNS Server to include in the DHCPv6 option data to clients using the nx_dhcpv6_create_dns_server service; now the server can be started with the

nx_dhcpv6_server_start service.

Note

All properties referenced in text are found in the NetX Duo DHCP IPv6 Server properties box, unless otherwise noted.

The server maintains a table of IPv6 addresses, updates their status, and indicates whether any addresses are leased out. For leased out addresses, the table indicates the client to which the address is leased. The size of this table is set by the Maximum Size of the Server's IP lease table property, and should be equal to or greater than the number of IPv6 addresses in the IPv6 address pool. The server maintains another table for client records; the size of this table is set by the Size of Server's Client record table property portion of the DHCP Server properties box and should be at least the size of the IPv6 lease table. If the server receives a Client Release or Decline, the server updates the IPv6 lease table and client record table, accordingly.

The DHCPv6 Server creates two timers. The first timer keeps track of the time remaining on IPv6 addresses leased to clients. The interval at which the server checks client leases is set by the Client lease time expiration check interval property, which defaults to 60 seconds. If the server issues leases with extremely short lease-expirations, that value should be reduced to approximately 10 or 20 percent of the timeout value. If a lease timeout expires, the lease is returned to the pool of IPv6 addresses and the client record is deleted. The second timer is used to monitor client-session inactivity; the default timeout for session inactivity timeout is 20 seconds. The interval in which the server checks its client records for an expired session inactivity timeout is the Interval for active session time update property in the NetX Duo DHCP IPv6 Common properties box, with the interval having a 3-second default time.

NetX Duo DHCP IPv6 Server Module Important Operational Notes and Limitations

NetX Duo DHCP IPv6 Server Module Operational Notes

- IPv6 and ICMPv6 must be enabled in NetX Duo. Verify that the NetX Duo IPv6 Support property is enabled in the NetX Duo Source properties box; this setting automatically enables ICMPv6.

If the underlying hardware supports ICMPv6 checksum computation, the following values can be left disabled:

- By default, the ICMPv6 checksum is disabled. To enable the checksum (assuming the hardware does not compute ICMPv6 checksums), locate the Checksum computation support on transmitted ICMPv6 packets property in the NetX Duo Source properties box, and set it to enabled. The Checksum computation support on received ICMPv6 packets property should also be enabled.
- Duplicate Address Detection (DAD) is recommended to verify the uniqueness of the server's global IPv6 address. This protocol is similar to sending gratuitous ARP probes in IPv4 to determine the uniqueness of an IPv4 address, but it is only applicable to IPv6 addresses. To enable DAD (which is disabled by default), set the Duplicate Address Detection support property to enabled in the NetX Duo Source properties box. The number of solicitation packets sent out for DAD is set by the Neighbor Solicitation message count before interface address marked valid property, which by default is 3 (and sent about a second apart). With this configuration, the application thread should wait about 4 seconds to let the DAD protocol complete.
- The following DHCPv6 parameters are supplied in DHCPv6 Server responses to the client:
    - T1 time: when the client should begin renewing its IPv6 address lease, is set by the Server interval for first client IP address renewal attempt property.
    - Preferred time: when the client IPv6 address is deprecated, is set by the Time

interval after which the client IP is deprecated property. This should be double the T1 time, as per RFC 3315 recommendations.
    - T2 time: when the client should begin rebinding an IPv6 address if renewing efforts failed, is set by the Server interval for the second client IP address renewal attempt property.
    - Valid lifetime: when the client IPv6 address is obsolete and should no longer be used by the client. This is set by the Time interval after which leased IP is invalid property, and should be double the preferred time, as per RFC 3315 recommendations.
    - There is no upper limit on the IPv6 lease time (valid lifetime), but the relative interval of these four time parameters must permit the logical order of renew and, if necessary, rebind state of the DHCPv6 protocol.
- The address-declined handler callback for handling a Client Decline message is not implemented on the current NetX Duo DHCPv6 Server. This callback is suggested by the RFC 3315 DHCPv6 specification for the server to notify the application of a declined address event.

NetX Duo DHCP IPv6 Server Module Limitations

- Rapid Commit option: optimizes the DHCPv6 address request process to just the Solicit and Reply message-exchange
- Reconfigure option: allows the server to initiate changes to the client's IP address status
- Unicast option: all client messages must be sent to All_DHCP_Relay_Agents_and_Servers multicast address rather than to the DHCPv6 Server directly.
- Identity Association for the Temporary Addresses (IA_TA) option: a temporary IP address granted to a client
- Multiple IA (IPv6 addresses) option: per client request
- Relay host between DHCPv6 Client and Server: client and server must be on the same network
- NetX Duo DHCPv6 Server: directly supports only the DNS Server option request
- Prefix Delegation option: is not supported.
- Option request callback: intended for the application and determines which DHCPv6 options to support and which information to supply to the DHCPv6 Server in response to the client. However, the processing of this information into the DHCPv6 Server response is not implemented. This callback has no effect on DHCPv6 Server messages
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.17.4 Including the NetX Duo DHCP IPv6 Server Module in an Application

This section describes how to include either or both the NetX Duo DHCP IPv6 Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo DHCP IPv6 Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo DHCP IPv6 Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_dhcp_server0NetX Duo DHCP IPv6 Server	Threads	**New Stack> X-Ware> NetX Duo> Protocols> NetX Duo DHPC IPv6 Server**

When the NetX Duo DHCP IPv6 Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

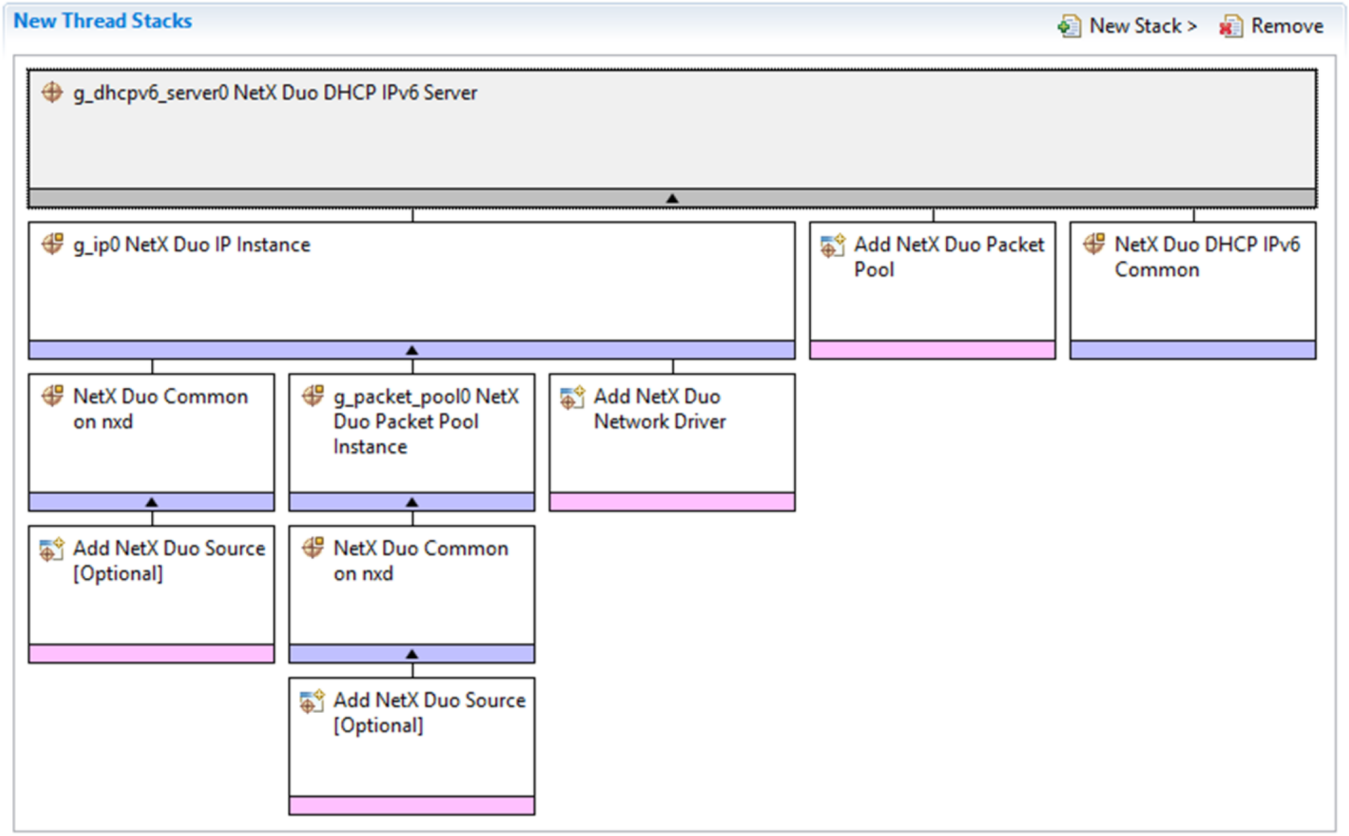


Figure 446: NetX Duo DHCP IPv6 Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp

- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.17.5 Configuring the NetX Duo DHCP IPv6 Server Module

The NetX Duo DHCP IPv6 Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo DHCP IPv6 Server Module

ISDE Property	Value	Description
Internal thread priority	1	Internal thread priority selection.
Client lease time expiration check interval (seconds)	60	Client lease time expiration check interval selection.
DHCPv6 packet receive timeout (seconds)	1	DHCPv6 packet receive timeout selection.
Server preference ranking for clients	0	Server preference ranking for clients selection.
Maximum options to extract from a client message	6	Maximum options to extract from a client message selection.
Server interval for first client IP address renewal attempt (seconds)	2000	Server interval for first client IP address renewal attempt selection.
Server interval for second client IP address renewal attempt (seconds)	3000	Server interval for second client IP address renewal attempt selection.
Time interval after which client IP is deprecated (seconds)	2*NX_DHCPV6_DEFAULT_T1_TIME	Time interval after which client IP is deprecated selection.
Time interval after which leased IP in invalid (seconds)	2*NX_DHCPV6_DEFAULT_PREFERRED_TIME	Time interval after which leased IP in invalid selection.
Maximum server status option message size (bytes)	100	Maximum server status option message size selection.
Maximum Size of the Server's IP lease table (count)	100	Maximum Size of the Server's IP lease table selection.

Size of the Server's Client record table (count)	120	Size of the Server's Client record table selection.
Server socket fragmentation option	Don't fragment, fragment okay Default: Don't fragment	Server socket fragmentation option selection.
Vendor assigned unique ID	abcdefghijklmnopqrstuvwxyz	Vendor assigned unique ID selection.
Private vendor ID	0x12345678	Private vendor ID selection.
Size of Vendor ID buffer (bytes)	48	Size of vendor ID buffer selection.
Client request success message: granted	IA OPTION GRANTED	Client request successmessage: granted selection.
Client request failure message: Failure unspecified	IA OPTION NOT GRANTED - FAILURE UNSPECIFIED	Client request failure message: Failure unspecified selection.
Client request failure message: No addresses available	IA OPTION NOT GRANTED - NO ADDRESSES AVAILABLE	Client request failure message: No addresses available selection.
Client request failure message: Invalid client request	IA OPTION NOT GRANTED - INVALID CLIENT REQUEST	Client request failure message: Invalid client request selection.
Client request failure message: Client not on link	IA OPTION NOT GRANTED - CLIENT NOT ON LINK	Client request failure message: Client not on link selection.
Client request failure message: Client must use multicast	IA OPTION NOT GRANTED - CLIENT MUST USE MULTICAST	Client request failure message: Client must use multicast selection.
Session inactivity timeout (seconds)	20	Session inactivity timeout selection.
Name	g_dhcpv6_server0	Module name.
Internal thread stack size (bytes)	4096	Internal thread stack size selection.
Name of address declined handler function	dhcpv6_address_declined_handler	Name of address declined handler function selection.
Name of option request handler	dhcpv6_option_request_handler	Name of option request handler selection.
Name of generated initialization function	dhcpv6_server_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it

might be useful to select different Ethernet interface pins and resets. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo DHCP IPv6 Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.

ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo DHCP IPv6 Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliabililty, Minimum cost Default: Normal	Type of service UDP requests selection.
Time to live	128	Time to live selection.
Packet Queue depth	5	Packet queue depth selection.
packet alocation timeout (seconds)	3	Packet allocation timeout selection.
Interval for active session time update (seconds)	3	Interval for active session time update selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G 2Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo DHCP IPv6 Server Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo DHCP IPv6 Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description

Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.17.6 Using the NetX Duo DHCP IPv6 Server Module in an Application

The following example assumes a system that is already established with a working and enabled IP, ARP and UDP, and the link is running.

The steps in using the NetX Duo DHCP IPv6 Server module in a typical application are:

1. Create a DUID for the DHCPv6 Server using the nx_dhcpv6_set_server_duid API.
2. Create a pool of assignable IPv6 addresses for the DHCPv6 Server using the nx_dhcpv6_create_ip_address_range API.
3. Set the DNS Server option for IPv6 using the nx_dhcpv6_create_dns_address API [Optional].
4. Start the DHCPv6 Server with the nx_dhcpv6_server_start API.
5. The DHCPv6 Server can be suspended as needed using thenx_dhcpv6_server_suspend API.
6. The DHCPv6 Server can be resumed using nx_dhcpv6_server_resume API [Optional].
7. Delete the DHCPv6 Server using the nx_dhcpv6_server_delete API.

The following figure illustrates common steps in a typical operational flow diagram:

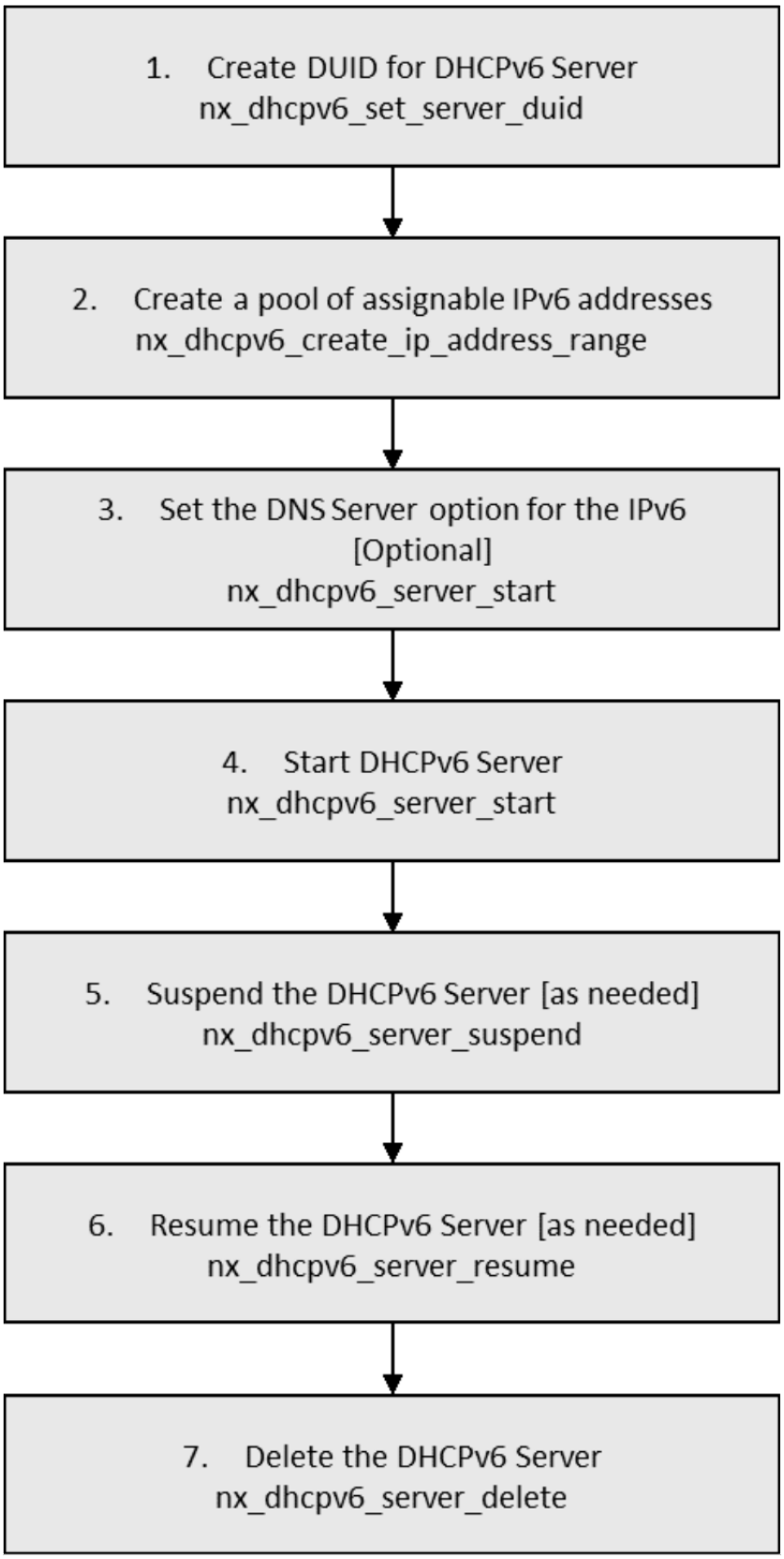


Figure 447: Flow Diagram of a Typical NetX Duo DHCP IPv6 Server Module Application

4.3.18 NetX/NetX Duo DNS Client

4.3.18.1 NetX/NetX Duo DNS Client Introduction

The Domain Name System (DNS) provides a distributed database that contains mapping between domain names and physical IP addresses. The database is referred to as "distributed" because there is no single entity on the Internet that contains the complete mapping. An entity that maintains a portion of the mapping is called a DNS server. The Internet is composed of numerous DNS servers, each of which contains a subset of the database. DNS servers also respond to DNS client requests for domain name mapping information, only if the server has the requested mapping. The DNS client protocol for NetX and NetX Duo provides the application with services to request mapping information from one or more DNS Servers.

Note

Except where noted, the NetX Duo DNS Client module is identical in the application, set-up and running of DNS queries as the NetX DNS Client module. For setting up the IP instance for IPv6 in NetX Duo, please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform.

Unsupported Features

Cache support has not been tested in this version of SSP.

IPV6 has not been tested for NetX Duo in this version of SSP.

NetX/NetX Duo DNS Client Module Features

- Optional creation of separate packet pool for DNS operations
- Support of Type A, AAAA, and NS DNS queries
- Support of CNAME, SRV, TXT, SOA, and MX DNS resource types
- Support for DNS cache for storing and retrieving cached DNS data
- High-level APIs for:
    - Name and IP address lookup
    - Adding and removing DNS servers
    - Creating and deleting the DNS instance
- NetX DNS is compliant with the following RFCs:
    - RFC1034 Domain Names – Concepts and Facilities
    - RFC1035 Domain Names - Implementation and Specification
    - RFC1480 The US Domain
    - RFC 2782 A DNS RR for specifying the location of services (DNS SRV)
    - RFC 3596 DNS Extensions to Support IP Version 6

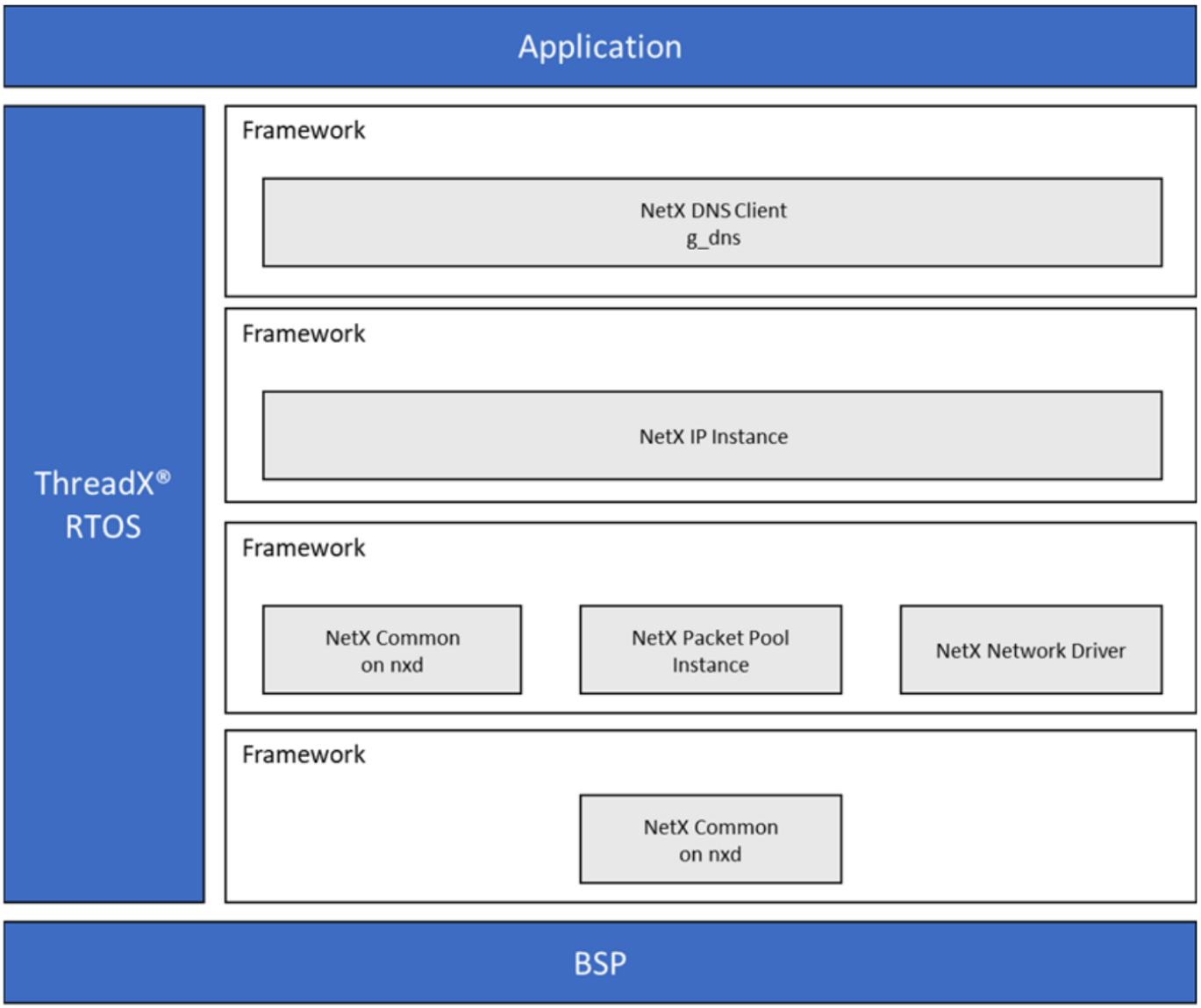


Figure 448: NetX/NetX Duo DNS Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo DNS Client Module in an Application for additional details.

4.3.18.2 NetX/NetX Duo DNS Client Module APIs Overview

The NetX/NetX Duo DNS Client module defines API functions for connecting, binding, listening, sending and receiving. A complete list of the available API functions, an example API function call and a short description of each can be found in the following table.

NetX/NetX Duo DNS Client Module API Summary

Function Name	Example API Call and Description
nx_dns_create	nx_dns_create(&my_dns, &my_ip, "My DNS"); Create a DNS Client instance.

nx_dns_delete	nx_dns_delete(&my_dns); Delete a DNS Client instance.
nx_dns_packet_pool_set	nx_dns_packet_pool_set(&my_dns, &packet_pool); Set the DNS Client packet pool.
nx_dns_get_serverlist_size	nx_dns_get_serverlist_size (&my_dns, 5); Return the size of the DNS Client's Server list.
nx_dns_info_by_name_get	nx_dns_info_by_name_get(&my_dns, "www.abc1234.com", ip_address, &port, 200); Return IPv4 address, port querying on input host name.
nx_dns_ipv4_address_by_name_get	nx_dns_ipv4_address_by_name_get(&client_dns, (UCHAR *)"www.my_example.com", record_buffer,sizeof(record_buffer),&record_count,500); Look up the IPv4 address for the specified host name.
nx_dns_host_by_address_get	nx_dns_host_by_address_get(&my_dns, IP_ADDRESS(192.2.2.10), &resolved_name[0], BUFFER_SIZE, 450); Look up a host name from a specified IP address (supports only IPv4 addresses).
nx_dns_host_by_name_get	nx_dns_host_by_name_get(&my_dns, "www.my_example.com", &ip_address, 4000); Look up an IP address from the host name (supports only IPv4 addresses).
nx_dns_server_add	nxd_dns_server_add(&my_dns, server_address); Add a DNS Server of the specified IP address to the Client's Server list (supports only IPv4).
nx_dns_server_get	nx_dns_server_get(&my_dns, 5, &my_server_address); Return the DNS Server in the Client list at the specified index (supports only IPv4 addresses).
nx_dns_server_remove	nx_dns_server_remove(&my_dns, IP_ADDRESS(202,2,2,13)); Remove a DNS Server from the Client list (supports only IPv4 addresses).
nx_dns_server_remove_all	nx_dns_server_remove_all(&my_dns); Remove all DNS Servers from the Client list.
nx_dns_cache_initialize	nx_dns_cache_initialize(&my_dns, &dns_cache, 2048); Initialize a DNS Cache.
nx_dns_cache_notify_clear	nx_dns_cache_notify_clear(&my_dns); Clear the DNS cache full notify function.

nx_dns_cache_notify_set	nx_dns_cache_notify_set(&my_dns, cache_full_notify_cb); Set the DNS cache full notify function.
nx_dns_cname_get	nx_dns_cname_get(&client_dns, (UCHAR *)"www.my_example.com ", record_buffer, sizeof(record_buffer), 500); Look up the canonical domain name for the input domain name alias.
nx_dns_authority_zone_start_get	nx_dns_authority_zone_start_get(&client_dns, (UCHAR *)"www.my_example.com", record _buffer, sizeof(record_buffer), &record_count, 500); Look up the start of a zone of authority associated with the specified host name.
nx_dns_domain_name_server_get	nx_dns_domain_name_server_get(&client_dns, (UCHAR *)" www.my_example.com ", record_buffer, sizeof(record_buffer), &record_count, 500); Look up the authoritative name servers for the input domain zone
nx_dns_domain_mail_exchange_get	nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR *)" www.my_example.com",record_buffer, sizeof(record_buffer), &record_count, 500); Look up the mail exchange associated with the specified host name.
nx_dns_domain_service_get	nx_dns_domain_service_get(&client_dns, (UCHAR *)"www.my_example.com ", record_buffer, sizeof(record_buffer), &record_count, 500); Look up the service(s) associated with the specified host name.
nxd_dns_ipv6_address_by_name_get**	nxd_dns_ipv6_address_by_name_get(&client_dns , (UCHAR *)"www.my_example.com", record__buffer, sizeof(record_buffer), &record_count, 500); Look up the IPv6 address from the specified host name.
nxd_dns_host_by_address_get**	nxd_dns_host_by_address_get(&my_dns, &host_address, resolved_name, sizeof(resolved_name), 4000); Look up a host name from the input IP address (supports both IPv4 and IPv6 addresses).
nxd_dns_host_by_name_get**	nxd_dns_host_by_name_get(&my_dns, "www.my_example.com", &ip_address, 4000, NX_IP_VERSION_V4); Look up an IP address from the input host name (supports both IPv4 and IPv6 addresses).

nxd_dns_server_add**	nxd_dns_server_add(&my_dns, &server_address); Add the input DNS Server to the Client server list (supports both IPv4 and IPv6 addresses).
nxd_dns_server_get**	nxd_dns_server_get(&my_dns, 5, &my_server_address); Return the DNS Server in the Client list at the specified index (supports both IPv4 and IPv6 addresses).
nxd_dns_server_remove**	nxd_dns_server_remove(&my_dns,& server_ADDRESS); Remove a DNS Server of the specified IP address from the Client list (supports both IPv4 and IPv6 addresses).

Note

Fordetails on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo DNS Client. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful.
NX_DNS_NO_SERVER	Client server list is empty.
NX_DNS_QUERY_FAILED	No valid DNS response received.
NX_DNS_NEED_MORE_RECORD_BUFFER	Input buffer size insufficient to hold the minimum data.
NX_PTR_ERROR*	Invalid IP or DNS pointer.
NX_CALLER_ERROR*	Invalid caller of this service.
NX_DNS_ERROR	Internal error in DNS Client processing.
NX_DNS_PARAM_ERROR	Invalid non-pointer input.
NX_DNS_CACHE_ERROR	Invalid Cache pointer.
NX_DNS_TIMEOUT	Timed out on obtaining DNS mutex.
NX_DNS_BAD_ADDRESS_ERROR	Null input address.
NX_DNS_INVALID_ADDRESS_TYPE	Index points to invalid address type (for example, IPv6).
NX_DNS_IPV6_NOT_SUPPORTED	Cannot process record with IPv6 disabled.
NX_NOT_ENABLED	Client not configured for this option.
NX_DNS_DUPLICATE_ENTRY	DNS Server is already in the Client list.
NX_NO_MORE_ENTRIES	No more DNS Servers allowed (list is full).

NX_DNS_SERVER_NOT_FOUND	Server not in client list.

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These error codes are only returned if error checking is enabled. For details on error-checking services, see *NetX User Guide for the Renesas Synergy™ Platform* or *NetX Duo User Guide for the Renesas Synergy™ Platform in NetX and NetX Duo*, respectively.

4.3.18.3 NetX/NetX Duo DNS Client Module Operational Overview

DNS Messages

The NetX DNS Client module creates the IP instance, enables UDP services in NetX, and initializes the network driver while registering a valid IP address. The module creates an UDP socket for sending and receiving DNS messages to DNS servers listening on port 53 for DNS queries.

To obtain a mapping, the DNS client prepares a DNS query message containing the name or the IP address to be resolved. The message goes to the first DNS server in the server list. If the server has a mapping, it replies to the DNS client using a DNS response message containing the requested mapping. If the server does not respond, the DNS client queries the next server on its list until all its DNS servers have been queried. If no response is received from all its DNS servers, the DNS client retransmits the DNS message starting back at the top of the server list. DNS queries are sent until the retransmission timeout expires. Until a response is received, each iteration down the list of servers doubles the retransmission timeout until the maximum transmission timeout is reached. This timeout is set in the *Maximum duration to retransmit a DNS query (seconds)* property. The default value is 64 seconds; the maximum number of times the DNS client iterates down the server list, which is set by the *Maximum retries for a server* property and defaults to 3.

The typical NetX DNS Client queries are:

- IPv4 address lookups (type A) by using the nx_dns_host_by_name_get service.
- Reverse lookups of IP addresses (type PTR queries) to obtain web host names using the nx_dns_host_by_address_get service.
- IPv6 address lookups (type AAAA) or IPv4 address lookups (type A), specified in the IP address data type input, in the nxd_dns_host_by_name_get service. (This is only available in the NetX Duo DNS Client.)
- Reverse lookups of IP addresses (type PTR queries) to obtain web-host names using the nxd_dns_host_by_address_get service. (This is only available in the NetX Duo DNS Client.)

The NetX Duo DNS Client module still supports nx_dns_host_by_name_get and nx_dns_host_by_address_get services. These are equivalent services, but they are limited to IPv4 network communication, so developers are encouraged to use the nxd_dns_host_by_name_get and nxd_dns_host_by_address_get services instead.

Extended DNS Resource Record Types

If the *Extended RR types support* property is enabled, the NetX DNS Client module also supports the following record type queries:

CNAME contains the canonical name for an alias

TXT contains a text string

NS contains an authoritative name server

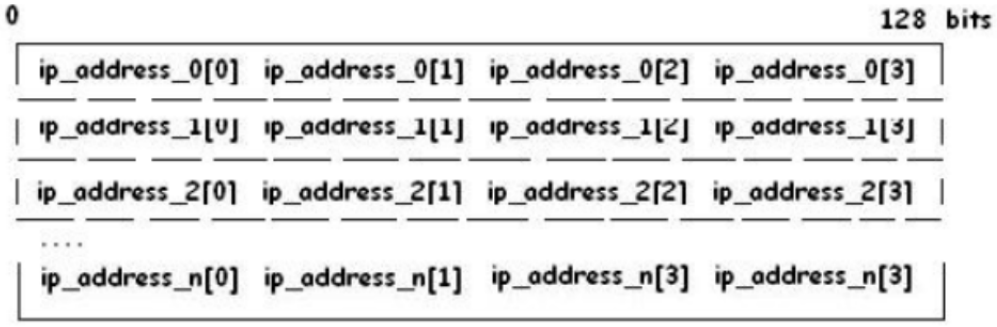
SOA contains the start of a zone of authority

MX used for mail exchange

SRV contains information on the service offered by the domain

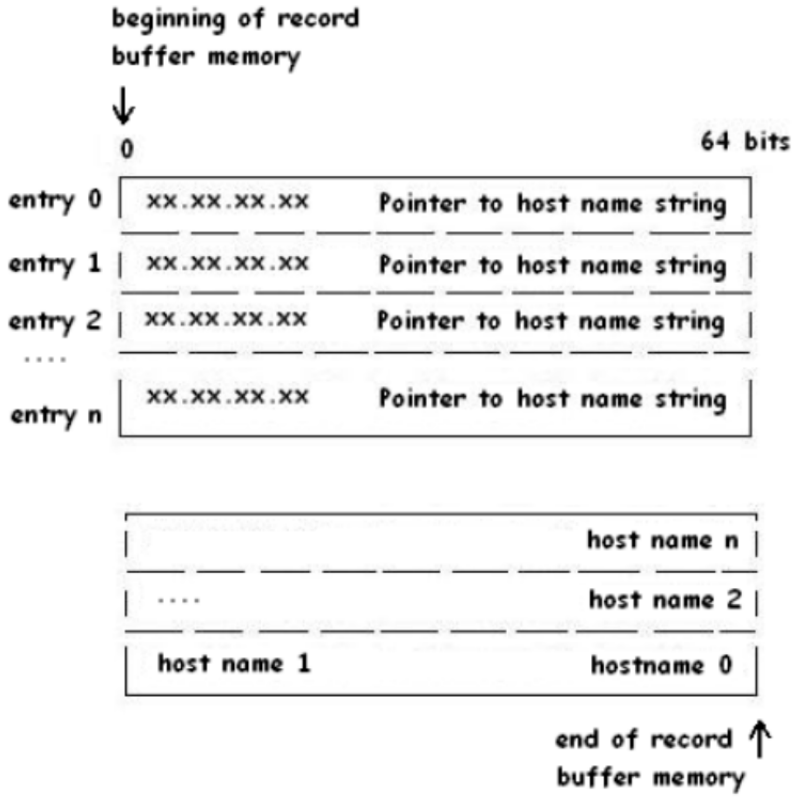
With the exception of CNAME and TXT record types, the application must supply a 4-byte aligned buffer to receive the DNS data record.

In the NetX DNS Client module, the record data is stored to make efficient use of buffer space. The following example shows a record buffer of fixed length (type AAAA record):



For queries whose record types have a variable data-length (such as NS records with variable-length host names), the NetX DNS Client module saves the data using the following methodology:

- Organizes the buffer supplied in the DNS Client query into fixed-length data and unstructured memory areas.
- Organizes the top of the memory buffer into 4-byte aligned record entries.
- For each record entry, holds the IP address and a pointer to the variable-length data for that IP address.
- Stores variable-length data for each IP address in unstructured area memory starting at the end of the memory buffer.
- Saves variable-length data for each successive record entry in the next area of memory, adjacent to the previous record entries' variable data.
- 'Grows' the variable data towards the structured area of memory holding record entries until there is insufficient memory to store another record entry and variable data (see following figure).



The NetX DNS Client queries use the record storage format to return the number of records saved to the record buffer; this information enables the application to extract NS records from the record buffer. After calling the nx_dns_domain_service_get with the receive_buffer pointer to the storage space, the application extracts the SRV data sequentially, taking advantage of the known size of the SRV entry:

```
NX_DNS_SRV_ENTRY *nx_dns_srv_entry_ptr[RECORD_COUNT];
status = nx_dns_domain_service_get(&client_dns,
                   (UCHAR *)"www.my_example.com",
                   &record_buffer[0], BUFFER_SIZE, &record_count,
                   NX_IP_PERIODIC_RATE);
for(i =0; i< record_count; i++)
{
    nx_dns_srv_entry_ptr[i] =
            (NX_DNS_SRV_ENTRY *)(record_buffer + i * sizeof(NX_DNS_SRV_ENTRY));
}
```

DNS Cache

If the *Cache support* property is enabled, the NetX DNS Client module supports the DNS cache feature. The application must set up the DNS cache using the nx_dns_cache_initialize service. When caching is enabled, the DNS client checks the DNS cache resource records before sending a DNS query. If it finds the answer in the cache, it directly returns it to the application. Otherwise, it sends out a query message to a DNS server and waits for the reply. When the DNS client gets the response message, it adds a resource record to the cache, if there is a cache entry available.

Each cache entry is a data structure used to hold a resource record. String entries (resource record name and data) in resource records are variable length, and as such, are not stored directly in the resource record. Instead, the resource record sets a pointer to the actual memory location in the cache where the strings are stored. Strings and resource records share the cache. Records are stored from the beginning of the cache and grow towards the end of the cache. String entries start from the end of the cache and grow towards the beginning of the cache. Each string entry has a length field and a counter field. When a string entry is added to the cache, and the same string is already present in the table, the counter value is incremented and no memory is allocated for the string. The cache is considered full if no more resource records or new string entries can be added to the cache.

NetX/NetX Duo DNS Client Module Important Operational Notes and Limitations

NetX/NetX Duo DNS Client Module Operational Notes

The NetX DNS Client requires a packet pool for transmitting DNS messages; by default, the application must set the packet pool before using DNS client services. This can either be the packet pool used by the IP instance (g_packet_pool0), or it can be a separate packet pool added to the project: **Azure RTOS**> **NetX Duo**> **NetX Duo Packet Pool Instance** (or **Azure RTOS**> **NetX**> **NetX Packet Pool Instance in NetX DNS Client**).

If the *Use application packet pool* is disabled, the DNS Client module creates the packet pool when the DNS client is created. The *Maximum DNS queries size* property determines the size of the packet payload and defaults to 512 bytes. The *Packets in DNS packet pool* property sets the number of packets in the DNS Client packet pool (defaults to 6).

Note

For user?created packet pools, the packet size holds the DNS maximum message size property. The default message size is 512 bytes, plus room for the UDP header (8 bytes), IPv4 header (20 bytes), or IPv6 header (40 bytes), and the network frame header. The Ethernet frame header is rounded up to 16 bytes for 4-byte alignment. This user-created packet pool is only used to send DNS packets. The IP packet pool is set to a 1568?byte packet payload. This packet pool setting works best for the Synergy network driver and the device MTU (typically, 1518 bytes), and it also avoids the need to chain packets in the driver layer.

Before sending DNS queries, the application must add one or more DNS servers to the server list kept by the DNS client. The maximum server list size is set by the *Maximum number of DNS Servers in the Client server list* property.

To add a DNS server, the application can use either the nx_dns_server_add service (limited to IPv4 packets) or the nxd_dns_server_add service supporting IPv4 and IPv6 packets.

Note

In the NetX Duo DNS Client, enabling the Client has DNS and Gateway Server property has no effect in the latest SSP release. When the DNS client is created, the IP instance does not have a router/gateway address set.

In the NetX DNS Client, enabling the *Client has Gateway Server* should be disabled. The IP instance

cannot have its gateway address by the time the DNS client is created and initialized by the configurator build process, and *DNS Client IP instance gateway* should not be used. To set a DNS server, it is recommended to use the nx_dns_server_add.

- To set up the DNS cache, the application can use the nx_dns_cache_initialize service with the cache memory buffer pointing to a previously defined buffer. To be notified if the cache is full, use the nx_dns_cache_notify_set service. This callback can be 'disabled' by clearing the callback using the nx_dns_cache_notify_clear service.
- A useful feature is the *Clear previous DNS queries from queue* property; this property enables the DNS client to remove any old DNS server responses from the DNS client receive queue when looking for a response that matches the current query. This means older packets received from previous DNS queries are discarded to prevent the DNS client socket from overflowing and dropping valid packets.
- Refer to the following list for NetX DNS services and their NetX Duo DNS equivalents:

NetX DNS API Service (IPv4 only)	NetX Duo DNS API Service (IPv4 and IPv6 supporte)
nx_dns_host_by_name_get	nxd_dns_host_by_name_get
nx_dns_host_by_address_get	nxd_dns_host_by_address_get
nx_dns_server_get	nxd_dns_server_get
nx_dns_server_add	nxd_dns_server_add
nx_dns_server_remove	nxd_dns_server_remove

NetX/NetX Duo DNS Client Module Limitations

- The DNS client supports one DNS request at a time; threads attempting to make another DNS request are temporarily blocked until the previous DNS request is complete.
- The NetX DNS Client module does not use data from authoritative answers to forward additional DNS queries to other DNS servers.
- Refer to the latest *SSP Release Notes* for any additional operational limitations for this module.

4.3.18.4 Including the NetX/NetX Duo DNS Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo DNS Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo DNS Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo DNS Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dns0 NetX DNS Client	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetX DNS Client**

g_dns0 NetX Duo DNS Client	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo DNS Client**

When the NetX and/or NetX Duo DNS Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

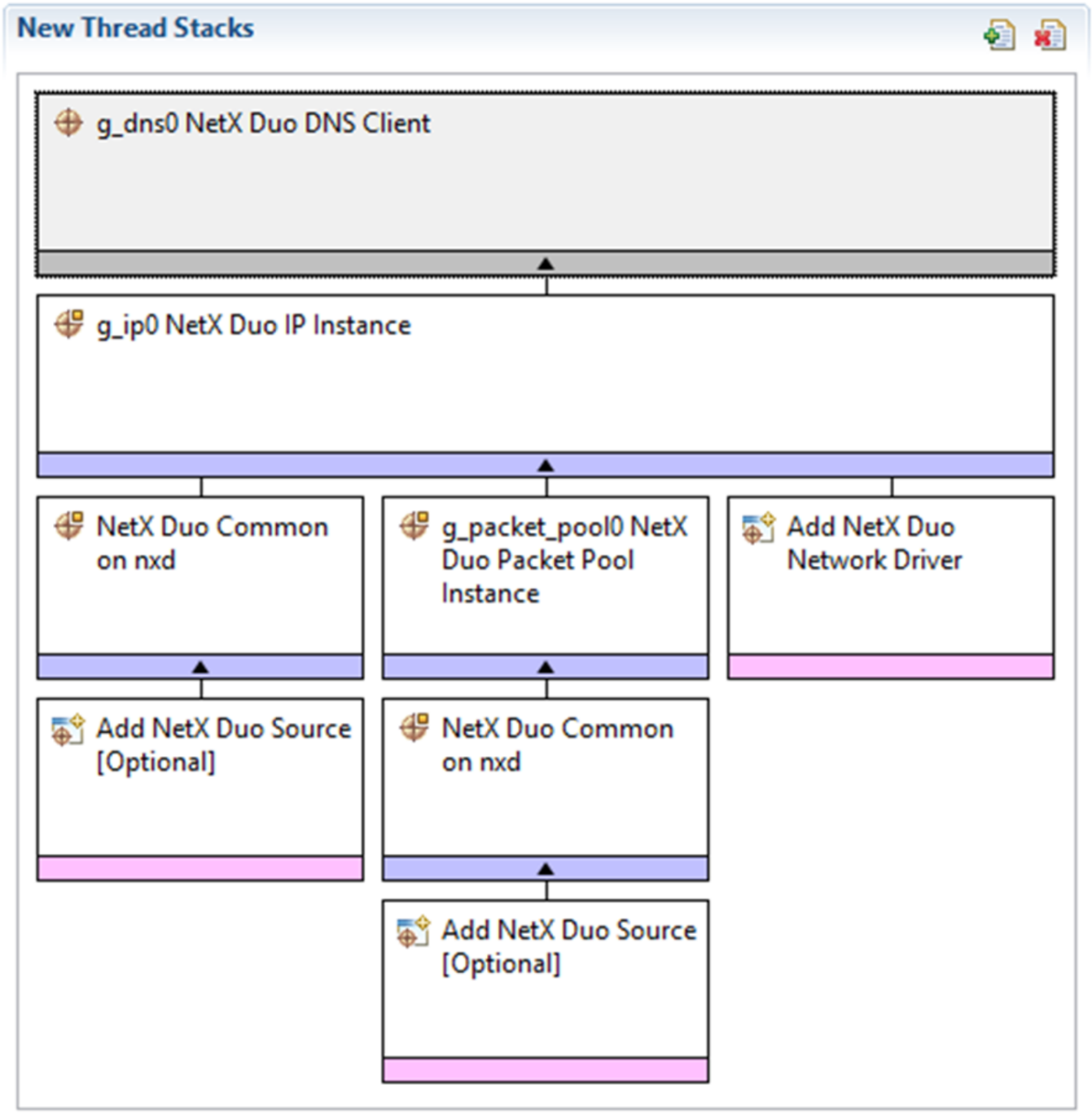


Figure 449: NetX/NetX Duo DNS Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp

- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.18.5 Configuring the NetX/NetX Duo DNS Client Module

The NetX/NetX Duo DNS Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo DNS Client Module

ISDE Property	Value	Description
DNS Control Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	DNS control type of service selection.
Socket fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Socket fragmentation option selection.
Time to live	128	Time to live selection.
**Client DNS IP version	IPv4, IPv6 Default: IPv4	Client DNS IP version selection.
Maximum number of DNS Servers in Client server list	5	Maximum number of DNS Servers in Client server list selection.
Maximum DNS queries size (bytes)	512	Maximum DNS queries size selection.
Packets in DNS packet pool (units)	16	Packets in DNS packet pool selection.
Maximum retries for a server	3	Maximum retries for a server selection.
Maximum duration to retransmit a DNS query (seconds)	64	Maximum duration to retransmit a DNS query selection.

Packet allocate timeout (seconds)	1	Packet allocate timeout selection.
Client has DNS and Gateway server	Enable, Disable Default: Disable	DNS control type of service selection.
DNS Client IP instance gateway (use commas for separation) (Not included in NetX Duo)	192,16,0,1	DNS client IP instance gateway selection.
Use application packet pool	Enable, Disable Default: Enable	Use application packet pool selection.
Clear previous DNS queries from queue	Enable, Disable Default: Disable	Clear previous DNS queries from queue selection.
Extended RR types support	Enable, Disable Default: Disable	Extended RR types support selection.
Cache support	Enable, Disable Default: Disable	Cache support selection.
Name	g_dns0	Module name.
Name of generated initialization function	dns_client_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC or IP Addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo DNS Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the

module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.

Auto Initialization	Enable, Disable Default: Enable	Auto initialization function.
Link status change callback	NULL	Link status change callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo DNS Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo DNS Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.18.6 Using the NetX/NetX Duo DNS Client Module in an Application

The steps in using the NetX/NetX Duo DNS Client module in a typical application are:

1. Wait for valid IP address using the nx_ip_status_check API.

2. Enable DNS caching by calling the nx_dns_cache_initialize [optional].

3. Add one or more servers to the client list using the nx_dns_server_add API.

4. Send DNS name query using nxd_dns_host_by_name_get API to obtain an IP address.

5. Extract DNS server resource records using known size and arrangement of records packaged by the DNS client.

The following figure illustrates common steps in a typical operational flow diagram:

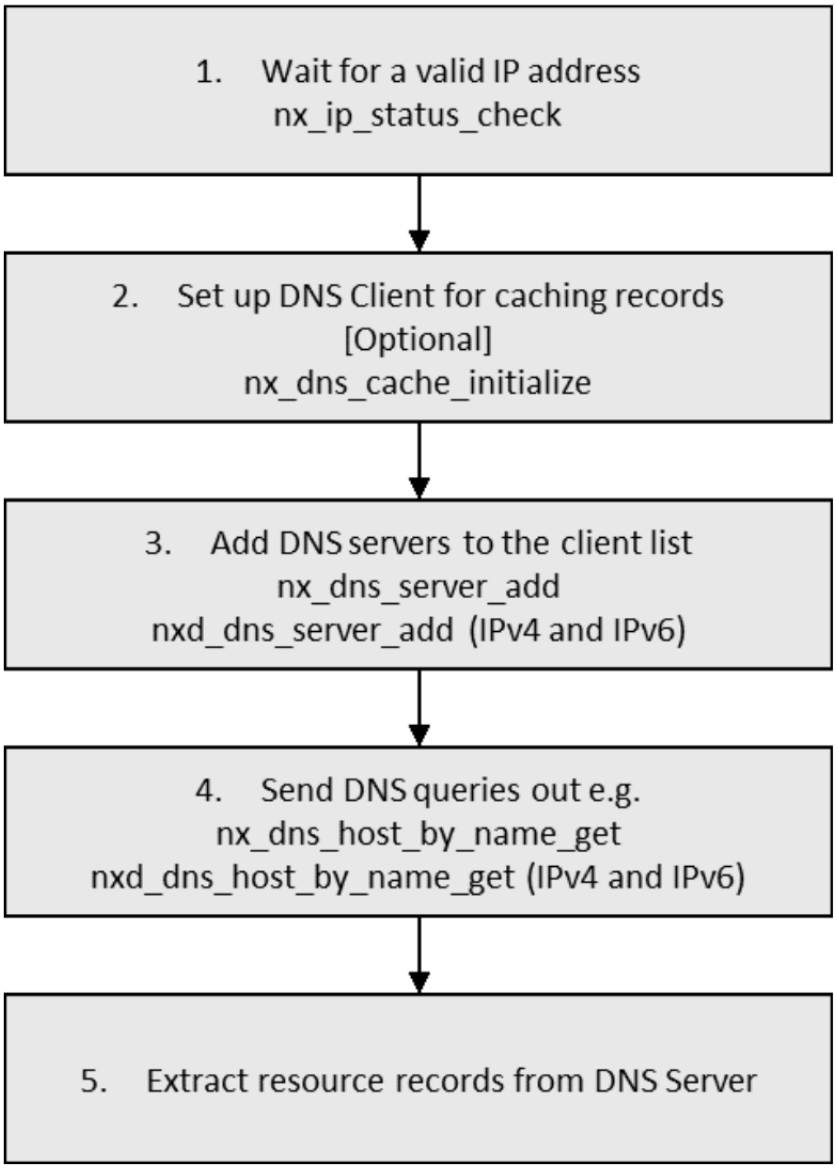


Figure 450: Flow Diagram of a Typical NetX/NetX Duo DNS Client Module Application

4.3.19 NetX/NetX Duo FTP Client

4.3.19.1 NetX/NetX Duo FTP Client Introduction

The File Transfer Protocol (FTP) is a protocol designed for file transfers. The FTP utilizes reliable Transmission Control Protocol (TCP) services to perform its file transfer function. The actual FTP file transfer is performed on a dedicated FTP connection.

Note

The NetX™ FTP Client module supports IPv4 only. The NetX Duo™ FTP Client module accommodates both IPv4 and IPv6 networks. IPv6 does not directly change the FTP protocol, although some changes in the original NetX FTP APIs are necessary to accommodate IPv6 and will be described in this document.

Unsupported Features

Multi-thread support has not been tested in this version of SSP.

Passive transfer mode has not been tested for NetX Duo in this version of SSP.

NetX/NetX Duo FTP Client Module Features

- Multi-thread support
  - High level APIs
  - Connecting and disconnecting
  - Directory operations
  - File operations

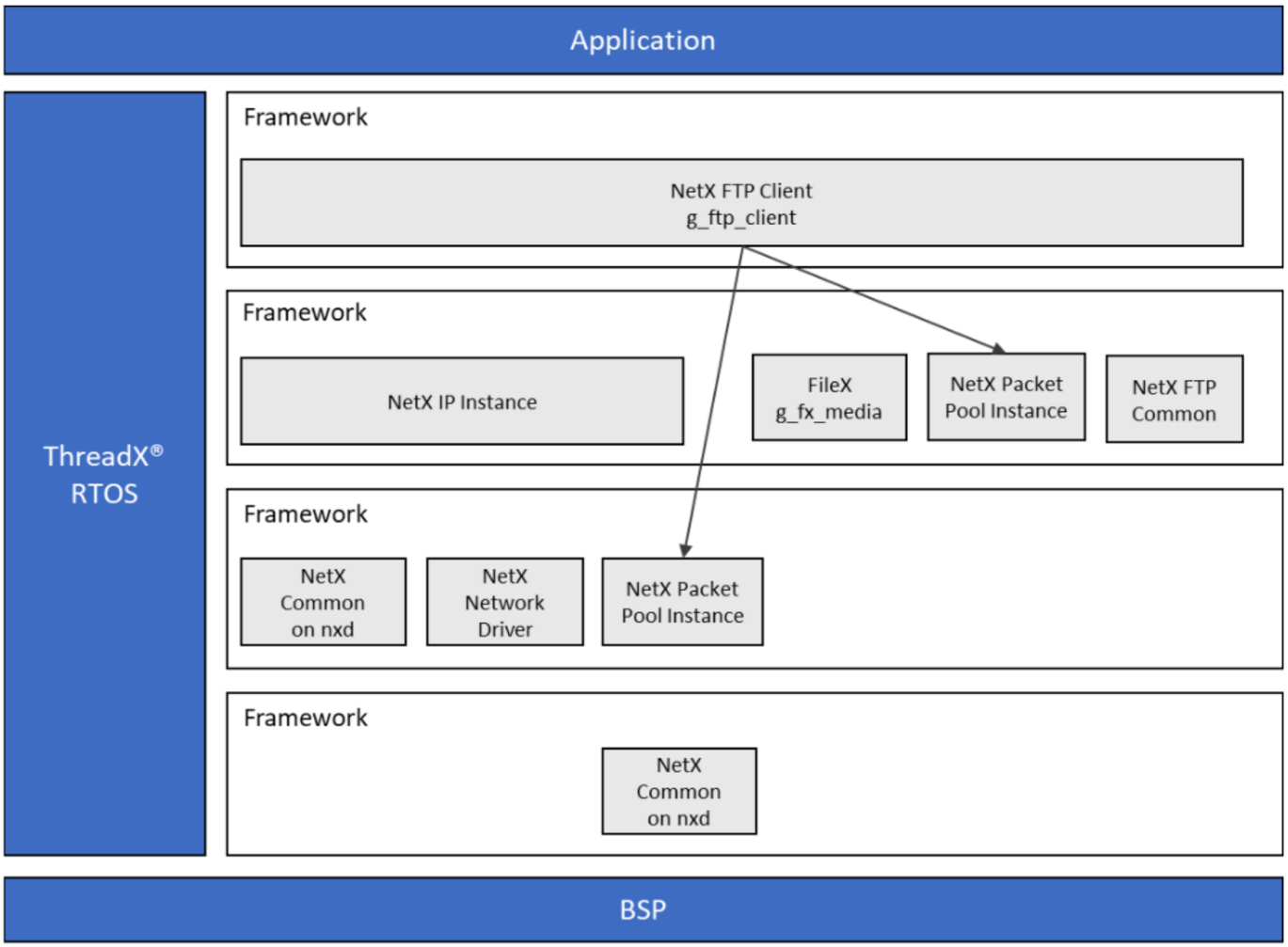


Figure 451: NetX/NetX Duo FTP Client Module Block Diagram

Note

In the figure above, the FileX and NetX (or NetX Duo) Network Driver modules have multiple implementation options available. See the descriptions just after the module stack figure in Including the NetX/NetX Duo FTP Client Module in an Application for additional details.

4.3.19.2 NetX/NetX Duo FTP Client Module APIs Overview

The NetX/NetX Duo FTP Client defines APIs for client creation/deletion, connecting/disconnecting to and from the server, directory operations and file operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo FTP Client Module API Summary

Function Name	Example API Call and Description
nx_ftp_client_create	nx_ftp_client_create(&my_client, "My Client", &client_ip, 2000, &client_pool); Delete an FTP Client instance.

nx_ftp_client_delete	nx_ftp_client_delete(&my_client); Delete an FTP Client instance.
nx_ftp_client_connect	nxd_ftp_client_connect(&my_client, server_ip_addr, NULL, NULL, 100); Connect to an FTP Server with IPv6 and IPv4 support.
**nxd_ftp_client_connect	nxd_ftp_client_connect(&my_client, server_ip_addr, NULL, NULL, 100); Connect to an FTP Server with IPv6 and IPv4 support.
nx_ftp_client_disconnect	nx_ftp_client_disconnect(&my_client, 200); Disconnect from the FTP Server.
nx_ftp_client_create	nx_ftp_client_directory_create(&my_client, "my_dir", 200); Create a directory on the server.
nx_ftp_client_directory_set	nx_ftp_client_directory_listing_set(&my_client, "my_dir", &my_packet, 200); Get a directory listing from the server.
nx_ftp_client_directory_delete	nx_ftp_client_directory_delete(&my_client, "my_dir", 200); Delete a directory on the server.
nx_ftp_client_directory_listing_get	nx_ftp_client_directory_listing_get(&my_client, "my_dir", &my_packet, 200); Get a directory listing from the server.
nx_ftp_client_directory_listing_continue	nx_ftp_client_directory_listing_continue(&my_client, &my_packet, 200); Continue a directory listing from the server.
nx_ftp_client_file_open	nx_ftp_client_file_open(&my_client, "my_file.txt", NX_FTP_OPEN_FOR_READ, 200); Open a file on the server.
nx_ftp_client_file_close	nx_ftp_client_file_close(&my_client, 200); Close the currently open file on the server.
nx_ftp_client_file_read	nx_ftp_client_file_read(&my_client, &my_packet, 200); Read from a file already open on the server.
nx_ftp_client_file_write	nx_ftp_client_file_write(&my_client, my_packet, 200); Write to an already open file on the server.
nx_ftp_client_file_rename	nx_ftp_client_file_rename(&my_client, "my_file.txt", "new_file.txt", 200); Rename a file on the server.

nx_ftp_client_file_delete	nx_ftp_client_file_delete(&my_client, "my_file.txt", 200); Delete a file on the server.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo FTP Client. For definitions of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	Successful FTP function.
NX_FTP_NOT_DISCONNECTED	FTP client is already connected.
NX_FTP_200_CODE_NOT_RECEIVED	Server rejects FTP connection.
NX_FTP_300_CODE_NOT_RECEIVED	Server rejects user/password.
NX_PTR_ERROR	Invalid FTP, username, or password pointer.
NX_CALLER_ERROR	Invalid caller of this service.
NX_IP_ADDRESS_ERROR	Invalid IP address.
NX_FTP_NO_2XX_RESPONSE_XXD	FTP server error response.
NX_FTP_NO_2XX_RESPONSE_PORT	FTP server response to PORT.
NX_FTP_NO_1XX_RESPONSE	FTP server response to NLST.
NX_FTP_END_OF_LISTING	No more entries in directory.
NX_FTP_NO_2XX_RESPONSE_DISCONNECT	FTP server response to disconnect.

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.19.3 NetX/NetX Duo FTP Client Module Operational Overview

FTP Client Requirements

To function properly, the NetX FTP Client package requires NetX. Similarly, the NetX Duo FTP Client package relies on NetX Duo. The host application must create an IP instance for running NetX services and periodic tasks. If running the FTP host application over an IPv6 network, IPv6, and ICMPv6 must be enabled on the IP task. TCP must be also enabled for either IPv6 or IPv4 networks. The IPv6 host application must set its link local and global IPv6 address using the IPv6 API and/or DHCPv6.

The FTP Server and Client are also designed to work with the FileX® embedded file system. If FileX is not available, the host developer can implement or substitute their own file system along the guidelines suggested in filex_stub.h by defining each of the services listed in that file.

The FTP Client portion of the NetX FTP package has no further requirements.

FTP File Names

FTP file names should be in the format of the target file system, usually FileX. They should be NULL terminated ASCII strings, with full path information if necessary. There is no specified limit for the size of FTP file names in the NetX FTP implementation. The packet pool payload size should be able to accommodate the maximum path and file name.

FTP Client Commands

The FTP has a simple mechanism for opening connections and performing file and directory operations. There is basically a set of standard FTP commands that are issued by the Client after a connection has been successfully established on the TCP well-known port 21. The following shows some of the basic FTP commands. Note that the only difference when FTP runs over IPv6 is that the PORT command is replaced with the EPRT command:

FTP Client Commands

FTP Command	Meaning
CWD path	Change working directory.
DELE filename	Delete specified file name.
EPRT ip_address, port	Provide IPv6 address and Client data port.
LIST directory	Get directory listing.
MKD directory	Make new directory.
NLST directory	Get directory listing.
NOOP	No operation, returns success.
PASS password	Provide password for login.
PORT ip_address,port	Provide IP address and Client data port.
PWD path	Pickup current directory path.
QUIT	Terminate Client connection.
RETR filename	Read specified file.
RMD directory	Delete specified directory.
RNFR oldfilename	Specify file to rename.
RNTO newfilename	Rename file to supplied file name.
STOR filename	Write specified file.
TYPE I	Select binary file image.
USER username	Provide username for login.

Note

These ASCII commands are used internally by the NetX FTP Client software to perform FTP operations with the FTP Server.

FTP Server Responses

Once the FTP Server processes the Client request, it returns a 3-digit coded response in ASCII followed by optional ASCII text. The numeric response is used by the FTP Client software to determine whether the operation succeeded or failed. The following list shows various FTP Server responses to Client requests:

First Numeric Field

First Numeric Field	Meaning
1xx	Positive preliminary status – another reply coming.
2xx	Positive completion status.
3xx	Positive preliminary status – another command must be sent.
4xx	Temporary error condition.
5xx	Error condition.

Second Numeric Field

Second Numeric Field	Meaning
0xx	Syntax error in command.
1xx	Positive preliminary status – another reply coming.
2xx	Positive completion status.
3xx	Positive preliminary status – another command must be sent.
4xx	Temporary error condition.
5xx	Error condition.

FTP Write Requests:

1. Client issues TCP connect to Server port 21.
2. Server sends 220 response to signal success.
3. Client sends USER message with username.
4. Server sends 331 response to signal success.
5. Client sends PASS message with password.
6. Server sends 230 response to signal success.
7. Client sends TYPE I message for binary transfer.
8. Server sends 200 response to signal success.
9. IPv6 applications: Client sends EPRT message with IP address and port. IPv4 applications: Client sends PORT message with IP address and port.
10. Server sends 200 response to signal success.
11. Client sends STOR message with file name to write.
12. Server creates data socket and connects with client data port specified in the previous EPRT or PORT command.
13. Server sends 125 response to signal file write has started.

14. Client sends contents of file through the data connection. This process continues until file is completely transferred.
15. When finished, Client disconnects data connection.
16. Server sends 250 response to signal file write is successful.
17. Clients sends QUIT to terminate FTP connection.
18. Server sends 221 response to signal disconnect is successful.
19. Server disconnects FTP connection.

FTP Authentication

Whenever a FTP connection takes place, the Client must provide the Server with a username and password. Some FTP sites allow what is called Anonymous FTP, that allows FTP access without a specific username and password. For this type of connection, anonymous should be supplied for username and the password should be a complete e-mail address.

You are responsible for supplying the NetX FTP Client and the NetX Duo FTP Client with login and logout authentication routines. These are supplied during the nxd_ftp_server_create and nx_ftp_server_create services and called from the password processing. The difference between the two is the nxd_ftp_server_create input function pointers to login and logout authenticate functions expect the NetX Duo address type NXD_ADDRESS. This data type holds both IPv4 or IPv6 address formats, making this function the Duo service supporting both IPv4 and IPv6 networks. The nx_ftp_server_create input function pointers to login and logout authenticate functions expect ULONG IP address type. This function is limited to IPv4 networks. You are encouraged to use the Duo service whenever possible.

If the login function returns NX_SUCCESS, the connection is authenticated and the FTP operations are allowed. Otherwise, if the login function returns something other than NX_SUCCESS, the connection attempt is rejected.

NetX/NetX Duo FTP Client Module Important Operational Notes and Limitations

NetX/NetX Duo FTP Client Module Operational Notes

FTP Multi-Thread Support

- The NetX FTP Client services can be called from multiple threads simultaneously. However, read or write requests for a particular FTP Client instance should be done in sequence from the same thread.

RFCs

- NetX Duo FTP is compliant with RFC 959, RFC 2428 and related RFCs.

NetX/NetX Duo FTP Client Module Limitations

FTP Constraints

The FTP standard has many options regarding the representation of file data. NetX FTP does not implement switch options like ls – al. The NetX FTP Server and the NetX Duo FTP Server expect to receive requests and their arguments in a single packet rather than consecutive packets.

Like UNIX implementations, NetX FTP assumes the following file format constraints:

- File Type: Binary
- File Format: Nonprint Only

- File Structure: File Structure Only

4.3.19.4 Including the NetX/NetX Duo FTP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo FTP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo FTP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo FTP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dns0 NetX FTP Client	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetX FTP Client**
g_dns0 NetX Duo FTP Client	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo FTP Client**

When the NetX and/or NetX Duo FTP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

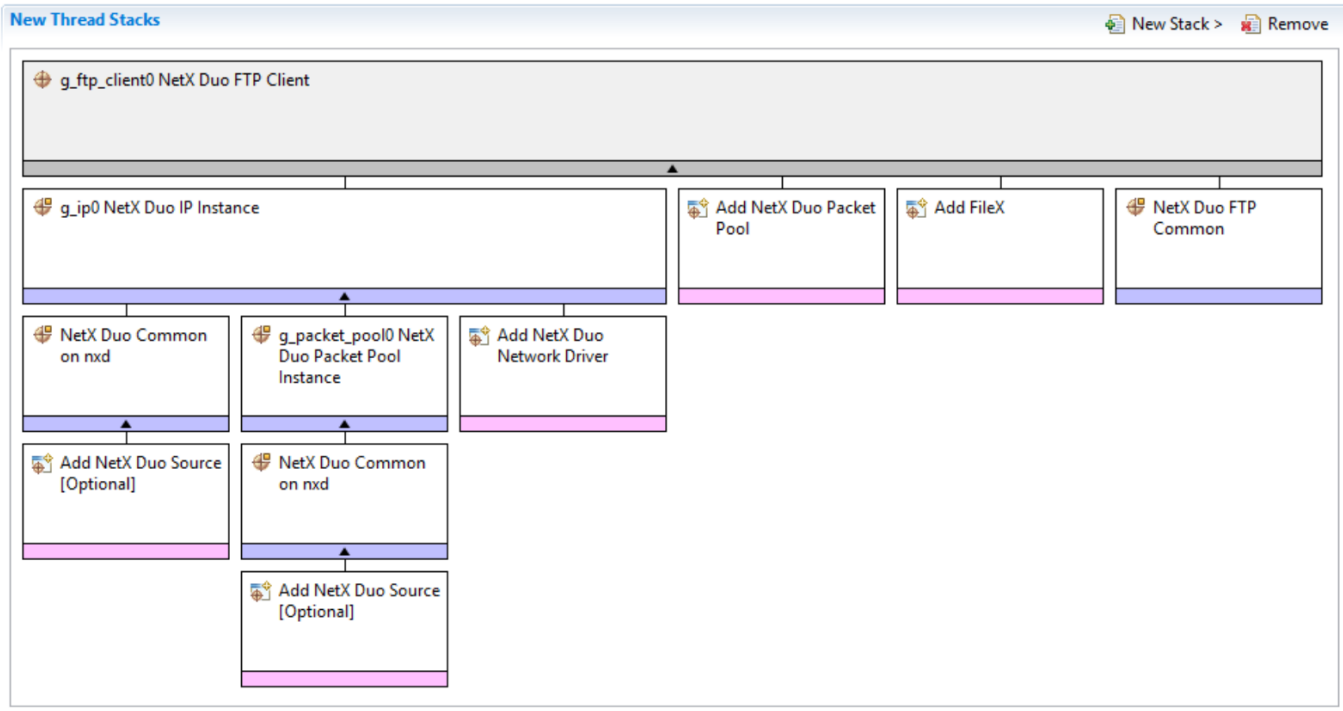


Figure 452: NetX/NetX Duo FTP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

Additionally, in the stack above, the FileX stack has also not been populated yet. There are multiple possible selections for the FileX module; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- FileX Stub
- FileX on Block Media (implemented on Block Media Framework on sf_block_media_ram)
- FileX on USB Mass Storage (implemented on USBX Host Class Mass Storage)

4.3.19.5 Configuring the NetX/NetX Duo FTP Client Module

The NetX/NetX Duo FTP Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous

manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo FTP Client Module

ISDE Property	Value	Description
**TCP socket to use	NX_ANY_PORT	TCP socket to use selection
Name	g_ftp_client0	Module name.
TCP socket window size (bytes)	2048	TCP socket window size selection
Name of generated initialization function	ftp_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different pins for the Ethernet peripheral. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo FTP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection.

Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function.
Link status change callback	NULL	Link status change callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo FTP Common Instance

ISDE Property	Value	Description
FileX support	Enabled, Disabled Default: Enabled	FileX support selection.
Control Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Control type of service selection.
Data Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Data type of service selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection.
Time to live	128	Time to live selection.
Duration between client inactivity check (seconds)	60	Duration between client inactivity check selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.

Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo FTP Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo FTP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.

TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.
MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.19.6 Using the NetX/NetX Duo FTP Client Module in an Application

The steps in using the NetX/NetX Duo FTP Client module in a typical application are:

Step 1. Create FTP Client using the nx_ftp_client_create API.

Step 2. Connect to the FTP Server using the nx_ftp_client_connect API.

The following steps can be used for writing to a file.

Step 3. Open a file using the nx_ftp_client_open API.

Step 4. Write to a file as needed using the nx_ftp_client_write API.

Step 5. Close a file using the nx_ftp_client_close API.

The following steps can be taken for reading from a file.

Step 6. Open a file using the nx_ftp_client_open API.

Step 7. Read from a file as needed using the nx_ftp_client_read API.

Step 8. Close a file using the nx_ftp_client_close API.

The following figure illustrates common steps in a typical operational flow diagram:

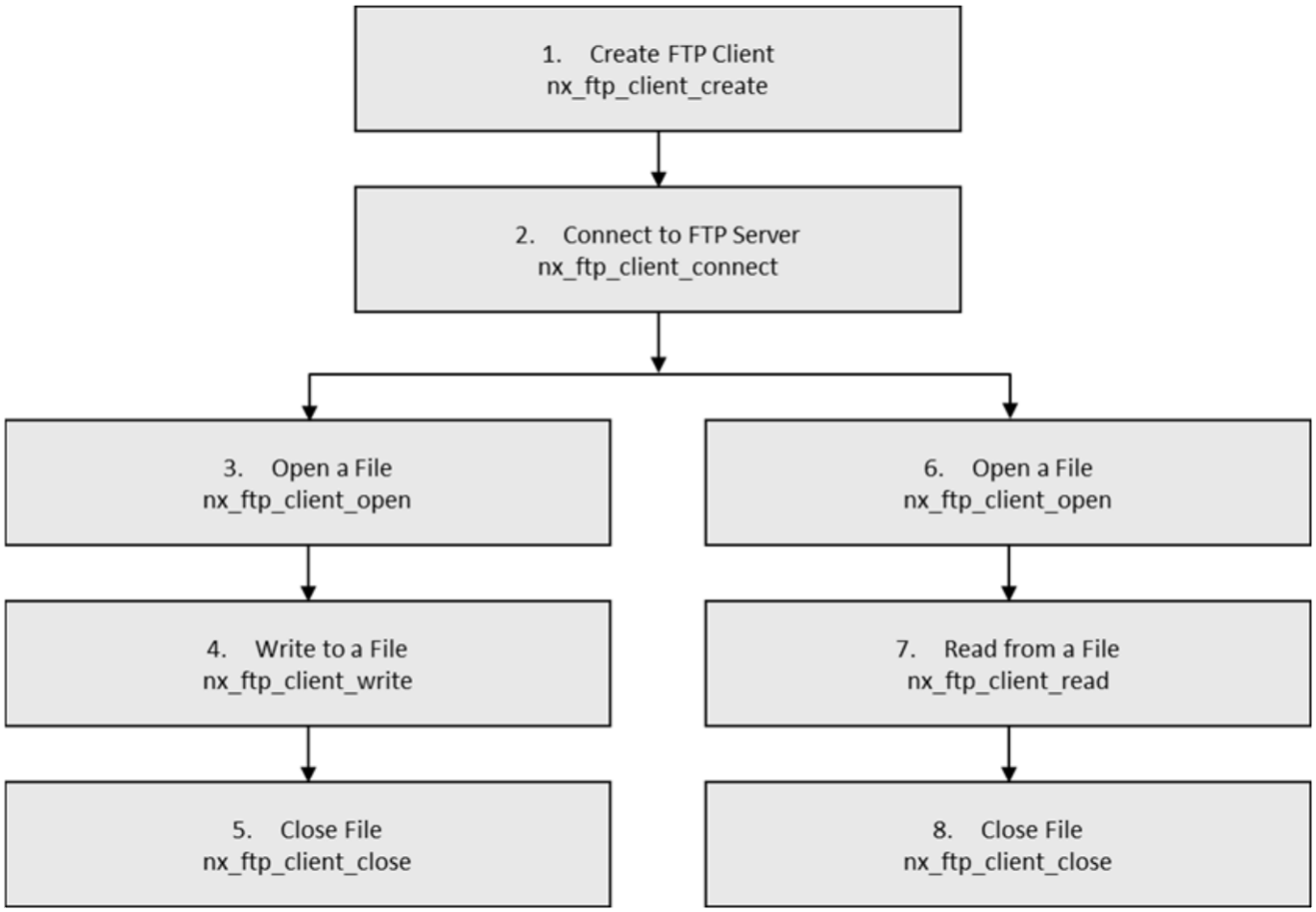


Figure 453: Flow Diagram of a Typical NetX/NetX Duo FTP Client Module Application

4.3.20 NetX/NetX Duo FTP Server

4.3.20.1 NetX/NetX Duo FTP Server Introduction

The File Transfer Protocol (FTP) is a protocol designed for file transfers. FTP utilizes reliable Transmission Control Protocol (TCP) services to perform its file transfer function. The actual FTP file transfer is performed on a dedicated FTP connection.

Note

The NetX™ FTP Server module is specific to IPv4. The NetX Duo™ FTP Server module accommodates both IPv4 and IPv6 networks. IPv6 does not directly change the FTP protocol, although some changes in the original NetX FTP APIs are necessary to accommodate IPv6 and will be described in this document.

Unsupported Features

Passive transfer mode has not been tested for NetX in this version of SSP.

NetX/NetX Duo FTP Server Module Features

- NetX is IPv4 specific
- NetX Duo supports both IPv4 and IPv6 networks
- Works with FileX® file system
- No limit to file name size; uses NULL terminated ASCII strings
- Supports TCP port 21 to field client requests
- Provide high-Level APIs for creating, starting, stopping and deleting service
- NetX FTP and NetX Duo FTP are compliant with RFC 959, RFC 2428 and related RFCs.

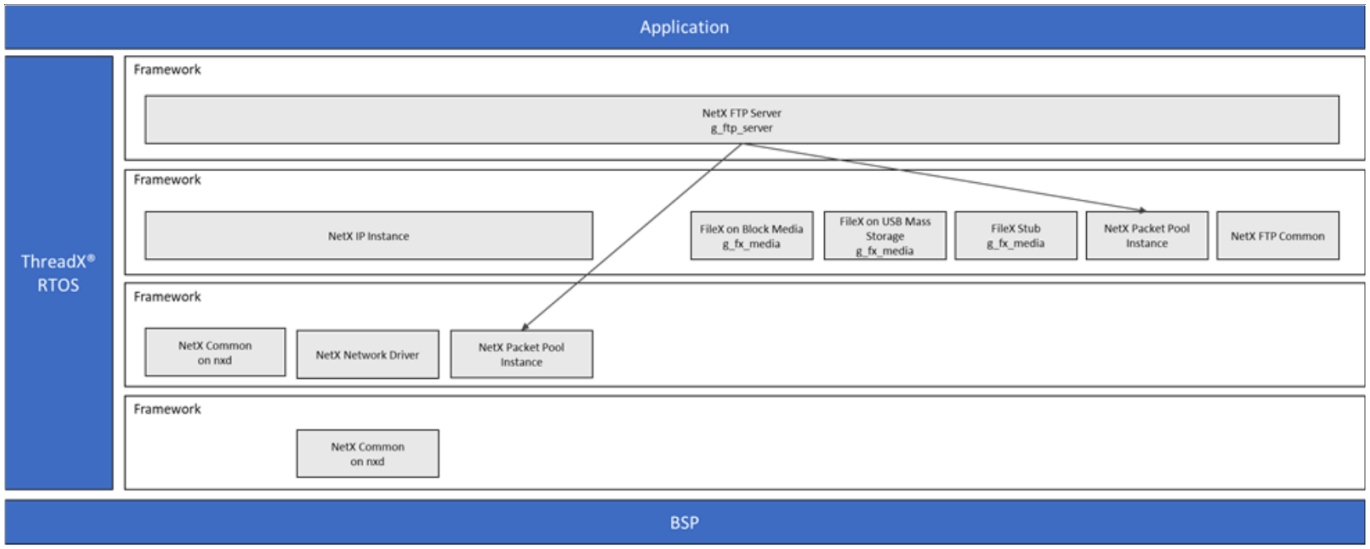


Figure 454: NetX/NetX Duo FTP Server Module Block Diagram

Note

In the figure above, the FileX and NetX (or NetX Duo) Network Driver modules have multiple implementation options available. See the descriptions just after the module stack figure in Including the NetX/NetX Duo FTP Server Module in an Application for additional details.

4.3.20.2 NetX/NetX Duo FTP Server Module APIs Overview

The NetX FTP Server defines APIs for creating, deleting, starting and stopping service. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo FTP Server Module API Summary

Function Name	Example API Call and Description
nx_ftp_server_create	nx_ftp_server_create(&my_server, "My Server Name", &ip_0, &ram_disk, stack_ptr, stack_size, &pool_0, my_login, my_logout); Create FTP Server with IPv4 support only.
**nxd_ftp_server_create	nxd_ftp_server_create(&my_server, "My Server Name", &ip_0, &ram_disk, stack_ptr, stack_size, &pool_0, my_duo_login, my_duo_logout); Create FTP Server with IPv4 and IPv6 support.
nx_ftp_server_delete	nx_ftp_server_delete(&my_server); Delete FTP Server.

nx_ftp_server_start	nx_ftp_server_start(&my_server); Start FTP Server.
nx_ftp_server_stop	nx_ftp_server_stop(&my_server); Stop FTP Server.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo FTP Client. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	Successful FTP server function.
NX_PTR_ERROR	Invalid FTP pointer.
NX_CALLER_ERROR	Invalid caller of this service.

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.20.3 NetX/NetX Duo FTP Server Module Operational Overview

Because FTP Client and Server operations are so closely linked, the following descriptions cover key elements of both Client and Server operations.

FTP Requirements

The NetX FTP package requires NetX or NetX Duo for proper operation. The host application must create an IP instance for running NetX services and periodic tasks. If running the FTP host application over an IPv6 network, IPv6, and ICMPv6 must be enabled on the IP task. TCP must be also enabled for either IPv6 or IPv4 networks. The IPv6 host application must set its linklocal and global IPv6 address using the IPv6 API and/or DHCPv6.

The FTP Server and Client are also designed to work with the FileX embedded file system. If FileX is not available, the host developer can implement or substitute their own file system along the guidelines suggested in filex_stub.h by defining each of the services listed in that file. This is discussed in later sections of this guide.

The FTP Client portion of the NetX FTP package has no further requirements.

The FTP Server portion of the NetX FTP package has several additional requirements. It requires complete access to TCP well-known port 21 for handling all Client FTP command requests and well-known port 20 for handling all Client FTP data transfers.

FTP File Names

FTP file names should be in the format of the target file system, usually FileX. They should be NULL terminated ASCII strings, with full path information if necessary. There is no specified limit for the

size of FTP file names in the NetX FTP implementation. The packet pool payload size should be able to accommodate the maximum path and/or file name.

FTP Client Commands

The FTP has a simple mechanism for opening connections and performing file and directory operations. There is a set of standard FTP commands that are issued by the Client after a connection has been successfully established on the TCP well-known port 21. The following table shows some of the basic FTP commands. Note that the only difference when FTP runs over IPv6 is that the PORT command is replaced with the EPRT command:

FTP Server Commands

FTP Command	Meaning
CWD path	Change working directory
DELE filename	Delete specified file name
EPRT ip_address, port	Provide IPv6 address and Client data port
LIST directory	Get directory listing
MKD directory	Make new directory
NLST directory	Get directory listing
NOOP	No operation, returns success
PASS password	Provide password for login
PORT ip_address,port	Provide IP address and Client data port
PWD path	Pickup current directory path
QUIT	Terminate Client connection
RETR filename	Read specified file
RMD directory	Delete specified directory
RNFR oldfilename	Specify file to rename
RNTO newfilename	Rename file to supplied file name
STOR filename	Write specified file
TYPE I	Select binary file image
USER username	Provide username for login

These ASCII commands are used internally by the NetX FTP Client software to perform FTP operations with the FTP Server.

FTP Server Responses

Once the FTP Server processes the Client request, it returns a 3-digit coded response in ASCII followed by optional ASCII text. The numeric response is used by the FTP Client software to determine whether the operation succeeded or failed. The following list shows various FTP Server responses to Client requests:

First Numeric Field

First Numeric Field	Meaning
1xx	Positive preliminary status – another reply coming.
2xx	Positive completion status.
3xx	Positive preliminary status – another command must be sent.
4xx	Temporary error condition.
5xx	Error condition.

Second Numeric Field

Second Numeric Field	Meaning
x0x	Syntax error in command.
x1x	Informational message.
x2x	Connection related.
x3x	Authentication related.
x4x	Unspecified.
x5x	File system related.

For example, a Client request to disconnect a FTP connection with the QUIT command will typically be responded with a 221 code from the Server – if the disconnect is successful.

FTP Communication

The FTP Server utilizes the well-known TCP port 21 to field Client requests. FTP Clients may use any available TCP port. The general sequence of FTP events are as follows. As mentioned previous, the only difference with FTP running over IPv6 is the PORT command is replaced with the EPRT command:

FTP Read File Requests:

1. Client issues TCP connect to Server port 21.
2. Server sends 220 response to signal success.
3. Client sends USER message with "username."
4. Server sends 331 response to signal success.
5. Client sends PASS message with "password."
6. Server sends 230 response to signal success.
7. Client sends TYPE I message for binary transfer.
8. Server sends 200 response to signal success.
9. Client sends PORT message with IP address and port.
10. Server sends 200 response to signal success.
11. Client sends RETR message with file name to read.
12. Server creates data socket and connects with client data port specified in the EPRT command.
13. Server sends 125 response to signal file read has started.

14. Server sends contents of file through the data connection. This process continues until file is completely transferred.
15. When finished, Server disconnects data connection.
16. Server sends 250 response to signal file read is successful.
17. Clients sends QUIT to terminate FTP connection.
18. Server sends 221 response to signal disconnect is successful.
19. Server disconnects FTP connection.

FTP Write Requests:

1. Client issues TCP connect to Server port 21.
2. Server sends 220 response to signal success.
3. Client sends USER message with "username."
4. Server sends 331 response to signal success.
5. Client sends PASS message with "password."
6. Server sends 230 response to signal success.
7. Client sends TYPE I message for binary transfer.
8. Server sends 200 response to signal success.
9. IPv6 applications: Client sends EPRT message with IP address and port. IPv4 applications: Client sends PORT message with IP address and port.
10. Server sends 200 response to signal success.
11. Client sends STOR message with file name to write.
12. Server creates data socket and connects with client data port specified in the previous EPRT or PORT command.
13. Server sends 125 response to signal file write has started.
14. Client sends contents of file through the data connection. This process continues until file is completely transferred.
15. When finished, Client disconnects data connection.
16. Server sends 250 response to signal file write is successful.
17. Clients sends QUIT to terminate FTP connection.
18. Server sends 221 response to signal disconnect is successful.
19. Server disconnects FTP connection.

FTP Authentication

Whenever a FTP connection takes place, the Client must provide the Server with a username and password. Some FTP sites allow what is called Anonymous FTP, which allows FTP access without a specific username and password. For this type of connection, **anonymous** should be supplied for username and the password should be a complete e-mail address.

You are responsible for supplying NetX FTP with login and logout authentication routines. These are supplied during the nxd_ftp_server_create and nx_ftp_server_create services and called from the password processing. The difference between the two is the nxd_ftp_server_create input function pointers to login and logout authenticate functions expect the NetX Duo address type NXD_ADDRESS. This data type holds both IPv4 or IPv6 address formats, making this function the duo service supporting both IPv4 and IPv6 networks. The nx_ftp_server_create input function pointers to login and logout authenticate functions expect ULONG IP address type. This function is limited to IPv4 networks. You are encouraged to use the duo service whenever possible.

If the login function returns NX_SUCCESS, the connection is authenticated and FTP operations are allowed. Otherwise, if the login function returns something other than NX_SUCCESS, the connection attempt is rejected.

NetX/NetX Duo FTP Server Module Important Operational Notes and Limitations

NetX/NetX Duo FTP Server Module Operational Notes

FTP Multi-Thread Support

- The NetX FTP Client services can be called from multiple threads simultaneously. However, read or write requests for a particular FTP Client instance should be done in sequence from the same thread.

RFCs

- NetX Duo FTP is compliant with RFC 959, RFC 2428 and related RFCs.

NetX/NetX Duo FTP Server Module Limitations

FTP Constraints

The FTP standard has many options regarding the representation of file data. NetX FTP does not implement switch options like ls – al. The NetX FTP Server and the NetX Duo FTP Server expect to receive requests and their arguments in a single packet rather than consecutive packets.

Like UNIX implementations, NetX FTP assumes the following file format constraints:

- File Type: Binary
- File Format: Nonprint Only
- File Structure: File Structure Only

4.3.20.4 Including the NetX/NetX Duo FTP Server Module in an Application

This section describes how to include either or both the NetX and NetX Duo FTP Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo FTP Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo FTP Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dns0 NetX FTPServer	Threads	**New Stack**> **X-Ware**> **NetX**> **Protocols**> **NetX FTPServer**
g_dns0 NetX Duo FTPServer	Threads	**New Stack**> **X-Ware**> **NetX Duo**> **Protocols**> **NetX Duo FTP Server**

When the NetX and/or NetX Duo FTP Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

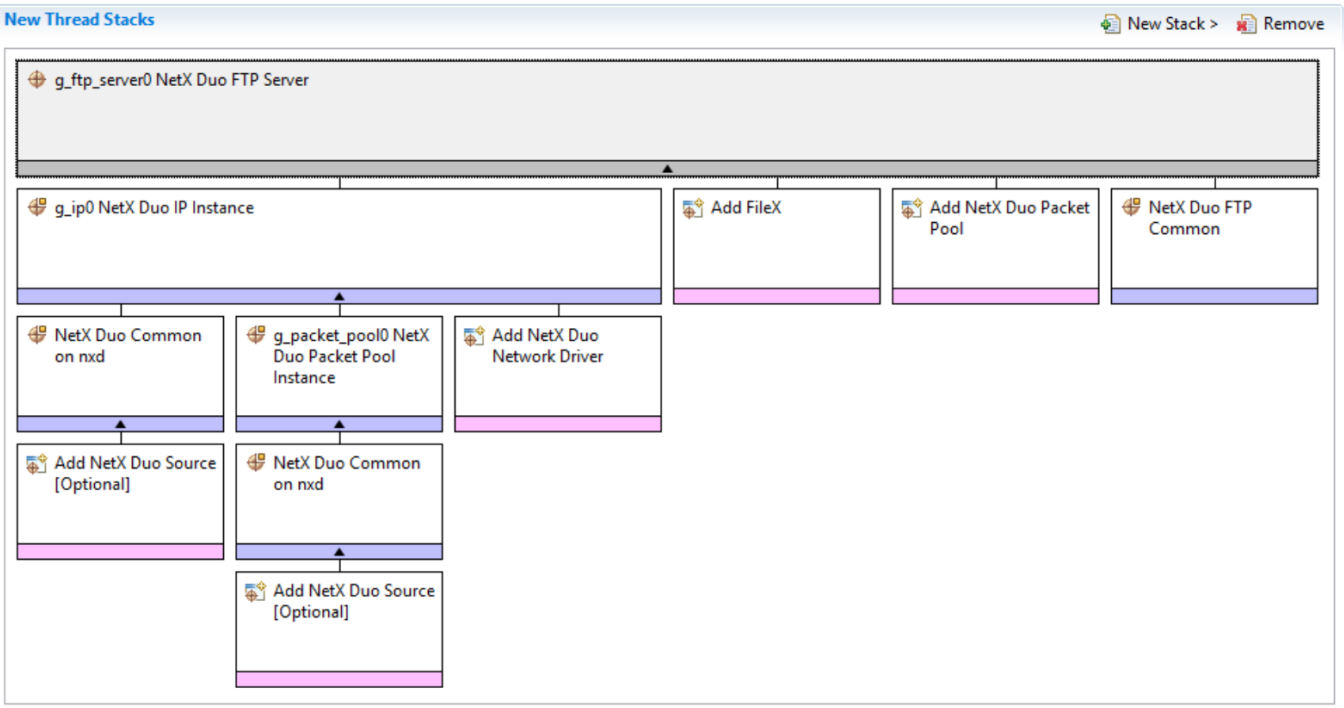


Figure 455: NetX/NetX Duo FTP Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

Additionally, in the stack above, the FileX stack has also not been populated yet. There are multiple possible selections for the FileX module; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- FileX Stub
- FileX on Block Media (implemented on Block Media Framework on sf_block_media_ram)
- FileX on USB Mass Storage (implemented on USBX Host Class Mass Storage)

4.3.20.5 Configuring the NetX/NetX Duo FTP Server Module

The NetX/NetX Duo FTP Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo FTP Server Module

ISDE Property	Value	Description
Internal Thread Priority	16	Internal thread priority selection.
Internal thread time slicing interval (ticks)	2	Internal thread time slicing interval selection.
Maximum clients to serve simultaneously	4	Maximum number of clients allowed.
Control window size (bytes)	400	Control window size selection.
Data window size (bytes)	2048	Data window size selection.
Duration internal services will suspend for (seconds)	1	Duration internal services will suspend for selection.
Maxixum username length (bytes)	20	Maximum username length selection.
Maximum password length (bytes)	20	Maximum password length selection.
Duration allowed with no activity (seconds)	240	Duration allowed with no activity.
Socket retransmit timeout (seconds)	2	Duration for initial timeout.
Maximum queued transmit packets	20	Maximum queued transmit selection.
Number of socket retransmissions	10	Maximum retries per selection.
Binary left shift as multiplier for retry duration	1	Binary left shift as multiplier for retry duration selection.
Name	g_ftp_server0	Module name.

Internal thread stack size (bytes)	4096	Internal thread stack size stacks selection.
Name of Login Function	ftp_login	Name of login function selection.
Name of Logout Function	ftp_logout	Name of logout function selection.
Name of generated initialization function	ftp_server_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC Addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo FTP Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name.
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection.
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection.
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection.
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection.
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection.

IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection.
IP Helper Thread Priority	3	IP Helper Thread Priority selection.
ARP	Enable	ARP selection.
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection.
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection.
TCP	Enable, Disable Default: Enable	TCP selection.
UDP	Enable, Disable Default: Enable	UDP selection.
ICMP	Enable, Disable Default: Enable	ICMP selection.
IGMP	Enable, Disable Default: Enable	IGMP selection.
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection.
Name of generated initialization function	ip_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function.
Link status change callback	NULL	Link status change callback selection.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo FTP Common Instance

ISDE Property	Value	Description
FileX support	Enabled, Disabled Default: Enabled	FileX support selection.

Control Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Control type of service selection.
Data Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Data type of service selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection.
Time to live	128	Time to live selection.
Duration between client inactivity check (seconds)	60	Duration between client inactivity check selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name.
Packet Size in Bytes	640	Packet size selection.
Number of Packets in Pool	16	Number of packets in pool selection.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo FTP Server Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo FTP Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select **Peripherals**> **Connectivity:ETHERC**> **ETHERC1.RMII**

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1.
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection.
REF50CK	P701	REF50CK pin.
TXD0	P700	TXD0 pin.
TXD1	P406	TXD1 pin.
TXD_EN	P405	TXD_EN pin.
RXD0	P702	RXD0 pin.
RXD1	P703	RXD1 pin.
RX_ER	P704	RX_ER pin.
CRS_DV	P705	CRS_DV pin.

MDC	P403	MDC pin.
MDIO	P404	MDIO pin.

Note

The example settings are for a project using the S7G2 Synergy MCU Group and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.20.6 Using the NetX/NetX Duo FTP Server Module in an Application

The steps in using the NetX/NetX Duo FTP Server module in a typical application are:

1. Create the FTP Server using the nx_ftp_server_create (or the nxd_ftp_server_create API for NetX Duo systems).

2. Start the FTP Server using the nx_ftp_server_start API.

3. Create the FTP Client using nx_ftp_client_create API.

4. Connect to the FTP Server using the nx_ftp_client_connect API.

5. Open a file using the nx_ftp_client_open API.

6. Read from a file using the nx_ftp_client_read API.

7. Close a file using the nx_ftp_client_close API.

The following figure illustrates common steps in a typical operational flow diagram:

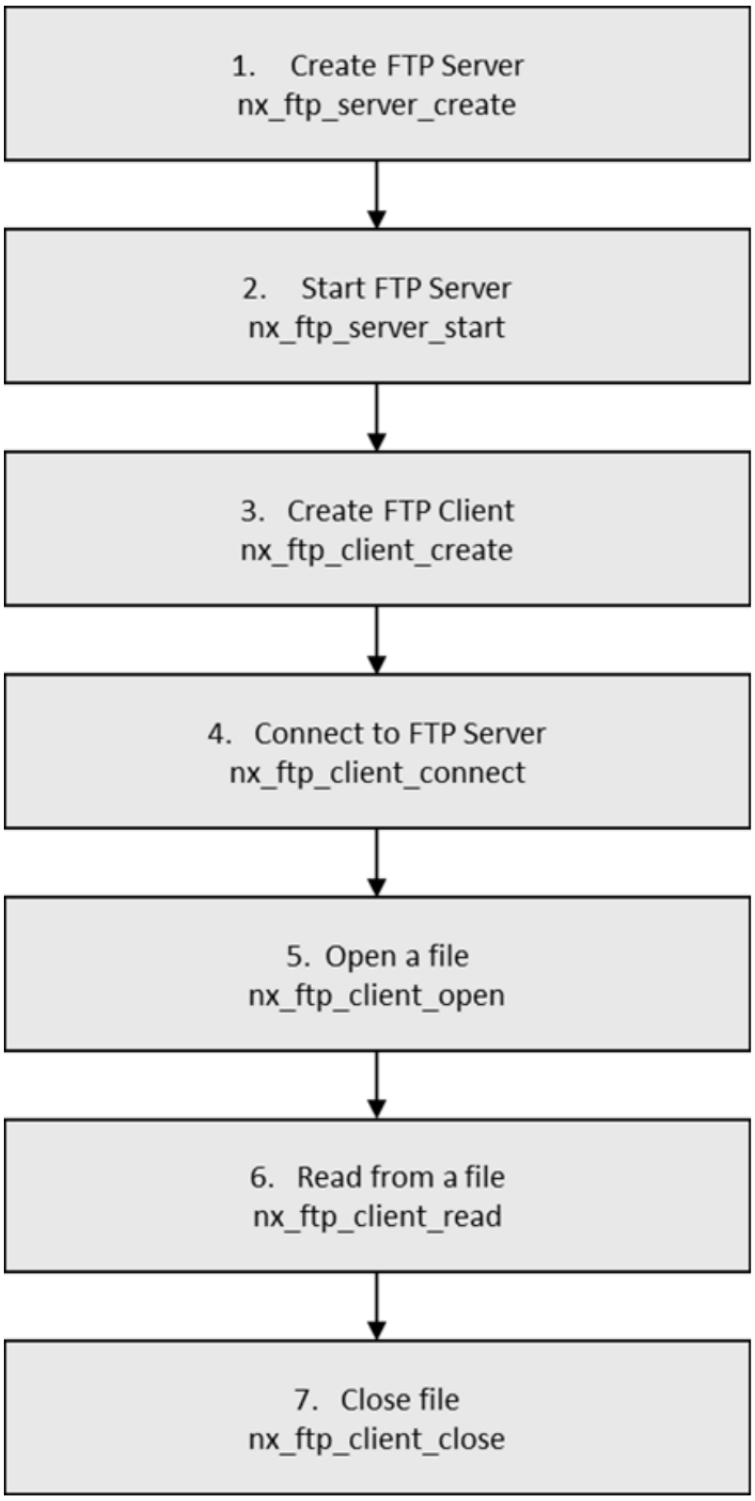


Figure 456: Flow Diagram of a Typical NetX/NetX Duo FTP Server Module Application

4.3.21 NetX/NetX Duo HTTP Client

4.3.21.1 NetX/NetX Duo HTTP Client Introduction

The Hypertext Transfer Protocol (HTTP) is a protocol designed for transferring content on the web. HTTP is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its content-transfer function. All operations on the web utilize the HTTP protocol.

Note

The NetX Duo™ HTTP Client accommodates both IPv4 and IPv6 networks while the NetX™ HTTP Client only supports IPv4 communications. IPv6 does not directly affect the HTTP protocol; some differences with the NetX HTTP Client are necessary to accommodate IPv6 and will be described in this document.

Unsupported Features

Multi-thread support has not been tested in this version of SSP. Multi-part support has not been tested in this version of SSP.

NetX/NetX Duo HTTP Client Module Features

- Provides high-level APIs to:
  - Create and delete an HTTP client instance
  - Send Get and Put requests to HTTP servers
- The NetX HTTP is compliant with RFC1945, Hypertext Transfer Protocol/1.0, RFC 2581, TCP Congestion Control, RFC 1122, Requirements for Internet Hosts and related RFCs.

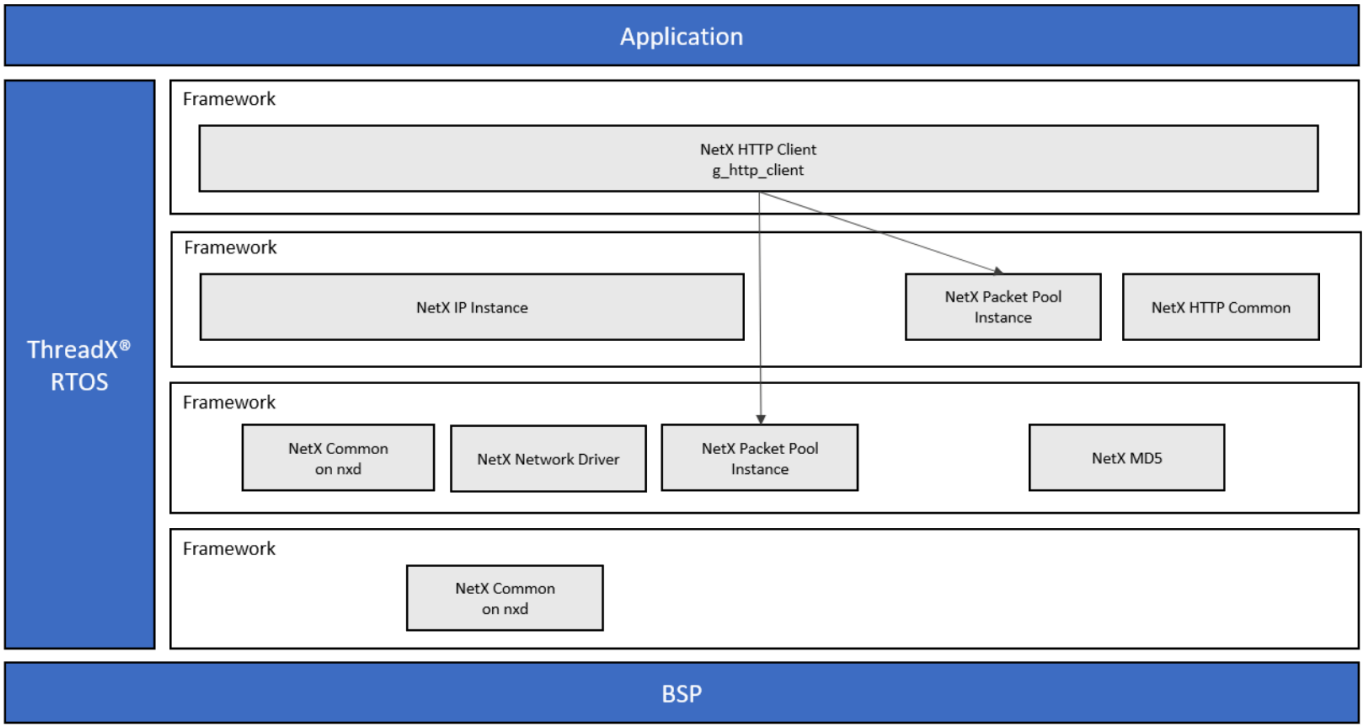


Figure 457: NetX/NetX Duo HTTP Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo HTTP Client Module in an Application for additional details.

4.3.21.2 NetX/NetX Duo HTTP Client Module APIs Overview

The NetX HTTP Client module defines APIs for creating, deleting, getting and putting. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo HTTP Client Module API Summary

Function Name	Example API Call and Description
nx_http_client_create	nx_http_client_create(&my_client, "my client", &ip_0, &pool_0, 100); Create an HTTP Client Instance.
nx_http_client_delete	nx_http_client_delete(&my_client); Delete an HTTP Client instance.
nx_http_client_get_start	nx_http_client_get_start(&my_client, IP_ADDRESS(1,2,3,5), "/TEST.HTM", NX_NULL, 0, "myname", "mypassword", 1000); Start an HTTP GET request (IPv4 only).
nxd_http_client_get_start**	nxd_http_client_get_start(&my_client, &server_ip_address, "/TEST.HTM", NX_NULL, 0, "myname", "mypassword", 1000); Start an HTTP GET request (IPv4 or IPv6)
nx_http_client_get_packet	nx_http_client_get_packet(&my_client, &next_packet, 1000); Get next resource data packet.
nx_http_client_put_start	nx_http_client_put_start(&my_client, IP_ADDRESS(1, 2, 3, 5), "/TEST.HTM", "myname", "mypassword", 20, NX_WAIT_FOREVER); Start an HTTP PUT request (IPv4 only).
nxd_http_client_put_start**	nxd_http_client_put_start(&my_client, &server_ip_address, "/client_test.htm", "name", "password", 103, 50); Start an HTTP PUT request (IPv4 or IPv6)
nx_http_client_put_packet	nx_http_client_put_packet(&my_client, packet_ptr, NX_WAIT_FOREVER); Send next resource data packet.
nx_http_client_set_connect_port**	nx_http_client_set_connect_port(&g_http_client0, 81); Connect to the HTTP server port on the specified port. Intended for situations where the Client must use another port beside 80.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo HTTP Client. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	Successful HTTP function
NX_CALLER_ERROR**	Invalid caller of the service
NX_PTR_ERROR**	Invalid HTTP, ip_ptr, or packet pool pointer
NX_INVALID_PORT**	Invalid port input
NX_HTTP_POOL_ERROR	Invalid payload size in packet pool
NX_HTTP_NOT_READY	HTTP Client not in ready state
NX_HTTP_PASSWORD_TOO_LONG	Password exceeded expected length
NX_HTTP_AUTHENTICATION_ERROR	Invalid name and/or password
NX_HTTP_FAILED	HTTP client error communicating with the HTTP server
NX_HTTP_GET_DONE	HTTP client get packet operation is complete
NX_HTTP_BAD_PACKET_LENGTH	Invalid packet received - length incorrect
NX_HTTP_INCOMPLETE_PUT_ERROR	Server responds before PUT is complete
NX_HTTP_REQUEST_UNSUCCESSFUL_CODE	Received an error code instead of 2xx from server
NX_HTTP_PASSWORD_TOO_LONG	Password exceeded expected length
NX_HTTP_USERNAME_TOO_LONG	Username exceeded expected length
NX_SIZE_ERROR	Invalid total size of resource in PUT request
NX_INVALID_PACKET	Invalid TCP packet; not enough room for packet header

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

**These are error codes which are only returned if error checking is enabled. Refer to the *NetX User's Guide* for the Renesas Synergy Platform or *NetX Duo User's Guide* for the Renesas Synergy Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.21.3 NetX/NetX Duo HTTP Client Module Operational Overview

The NetX HTTP Client module creates an IP instance which carries out NetX operations and enables TCP services in the NetX library. It creates the HTTP client instance and a TCP socket for sending and receiving HTTP messages to the server listening on port 80. The HTTP client requires a packet pool; the module can supply one by sharing the IP default packet pool (g_packet_pool0), or by creating a new one. The minimum packet payload is set by the Minimum packet size property of the HTTP client instance. This packet pool is used by the HTTP client only to transmit packets, so the packet-pool size and payload can be optimized based on the expected size and the number of HTTP client packets sent out.

The NetX Duo HTTP Client supports both IPv4 and IPv6 connections; if the HTTP client needs to use

IPv6 to connect to a server, make sure the NetX Duo IPv6 Support property is enabled in the NetX Duo Source element. It may also be necessary to enable ICMPv6 checksum computation for the underlying ICMPv6 protocols; to do so, set the Checksum computation support on transmitted ICMPv6 packets and Checksum computation support on received ICMPv6 packets properties of the NetX Duo source element to be enabled. (If the host hardware automatically computes ICMPv6 checksums, these can be left disabled.) Make sure the IPv6 Global Address of the Client host is set in the IP instance element; NetX Duo will do the necessary processing to enable IPv6 and ICMPv6 services required for IPv6 underlying protocols.

Once the HTTP client has a valid IP address, it can make PUT and GET requests. To upload packets, use the nx_http_client_put_start service. This service has a server IP address input, so the HTTP client can connect to the server. If the data to upload exceeds more than one packet, the application uses the nx_http_client_put_packet service until all the data is uploaded. To download data from the server, use the nx_http_client_get_start service; this requires the server IP address so the HTTP client can connect to the server. If the data to download exceeds more than one packet, the application uses the nx_http_client_get_packet service until all the data is downloaded; this is indicated by getting the NX_HTTP_GET_DONE status return.

In the NetX Duo HTTP Client module, the application can use the nxd_http_client_put_start for IPv4 connections, and the nxd_http_client_get_start service for either IPv4 or IPv6 connections; nx_http_client_put_start and nx_http_client_get_start services are also available in the Net Duo HTTP Client. The nx_http_client_put_packet and nx_http_client_get_packet services do not require an HTTP Server IP address, so there is no Duo-equivalent APIs for these services.

In NetX Duo HTTP Client, the nx_http_client_set_connect_port service is available for those circumstances where the HTTP client needs to connect to the HTTP server on a port other than the default of Port 80.

HTTP Server Responses

Once the HTTP server processes the client command, it returns an ASCII response string that includes a 3-digit numeric-status code listed in the following table. The numeric response is used by the HTTP client software to determine whether the operation succeeded or failed.

Various HTTP server responses to client commands

Numeric Field	Meaning
200	Request was successful
400	Request was not formed properly
401	Unauthorized request, client needs to send authentication
404	Specified resource in request was not found
500	Internal HTTP server error
501	Request not implemented by HTTP server
502	Service is not available

For example, a successful client request to PUT the file test.htm is responded to with the message HTTP/1.0 200 OK.

NetX/NetX Duo HTTP Client Module Important Operational Notes and Limitations

NetX/NetX Duo HTTP Client Module Operational Notes

- The HTTP client packet pool must be large enough to hold the complete HTTP header.
- The wait option for disconnecting from the server before deleting the client TCP socket is set by the Operation Timeout property; the wait option for all other HTTP client services is set in the API.
- Both GET and PUT start services require a resource, username and password as an input. The maximum size of each is set by the Maximum resource name length, Maximum username length and Maximum password length properties in the NetX HTTP Common element. When the GET and PUT operation is completed, the HTTP client disconnects from the server.
- The HTTP client TCP socket receive window is set by the TCP socket window size property. This is used in the TCP protocol for one peer to let the other know not to send more data pending-acknowledgment packets for data already received.

NetX/NetX Duo HTTP Client Module Limitations

- The HTTP protocol in NetX and NetX Duo implements the HTTP 1.0 standard; it does not support 1.1. The constraints are as follows:
  - Persistent connections are not supported
  - Request pipelining is not supported
  - Content compression is not supported
  - TRACE, OPTIONS, and CONNECT requests are not supported
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.21.4 Including the NetX/NetX Duo HTTP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo HTTP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo HTTP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo HTTP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dns0 NetXHTTP Client	Threads	New Stack> X-Ware> NetX> Protocols> NetXHTTP Client
g_dns0 NetX Duo HTTP Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo HTTP Client

When the NetX and/or NetX Duo HTTP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

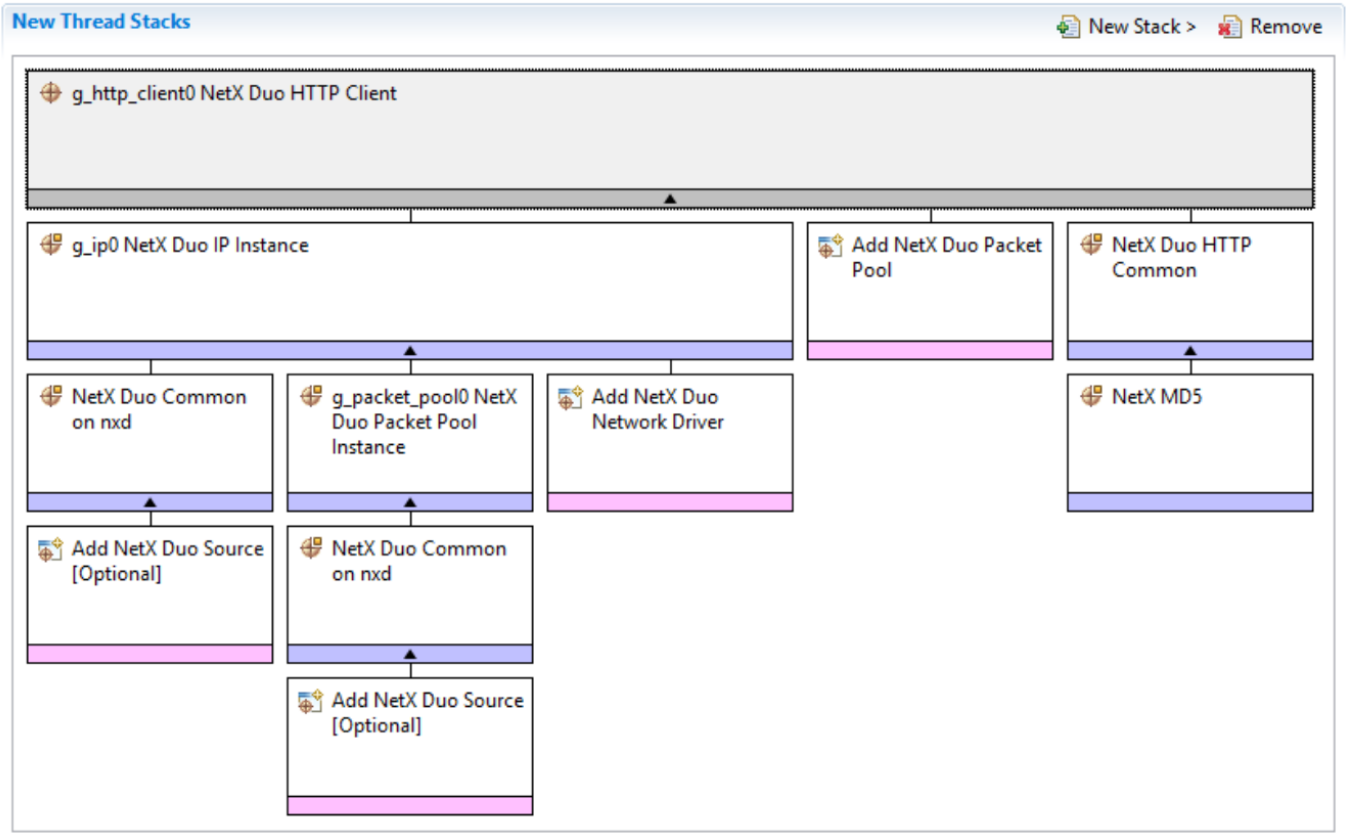


Figure 458: NetX/NetX Duo HTTP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.21.5 Configuring the NetX/NetX Duo HTTP Client Module

The NetX/NetX Duo HTTP Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes

and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo HTTP Client Module

ISDE Property	Value	Description
Minimum packet size (bytes)	300	Minimum packet size selection
Operation timeout (seconds)	10	Operation timeout selection
*Maximum password length (bytes)	20	Maximum password length selection
*Maximum username length (bytes)	20	Maximum username length selection
Name	g_http_client0	Module name
TCP socket window size (bytes)	1024	TCP socket window size selection
Name of generated initialization function	http_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

*Indicates properties that are not available in NetX Duo.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different pins for the Ethernet peripheral. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo HTTP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the

module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection

Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo HTTP Common Instance

ISDE Property	Value	Description
Type of Service	Normal, Minimum delay, Maximum data, Maximum reliabililty, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection
Time to live	128	Time to live selection
MD5 Support	Enable, Disable Default: Disable	MD5 support selection
Maximum resource name length (bytes)	40	Packet queue depth selection
**Maximum password length (bytes)	20	Maximum password length selection
**Maximum username length (bytes)	20	Minimum username length selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo HTTP Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo HTTP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.21.6 Using the NetX/NetX Duo HTTP Client Module in an Application

The steps in using the NetX/NetX Duo HTTP Client module in a typical application are:

1. Wait for valid IP address and network driver initialization using the nx_ip_status_check API.

2. Upload data to the HTTP server with the nx_http_client_put_start API. For IPv6 connections, use the nxd_http_client_put_start API in the NetX Duo HTTP Client (which can also be used for IPv4 connections).

3. Download data from the HTTP server with the nx_http_client_get_start API. For IPv6 connections,

use the nxd_http_client_get_start API in the NetX Duo HTTP Client (which can also be used for IPv4 connections).

4. Delete the HTTP client with the nx_http_client_delete API. (Note that the packet pool can also be deleted if it's not used elsewhere in the application.)

The following figure illustrates common steps in a typical operational flow diagram:

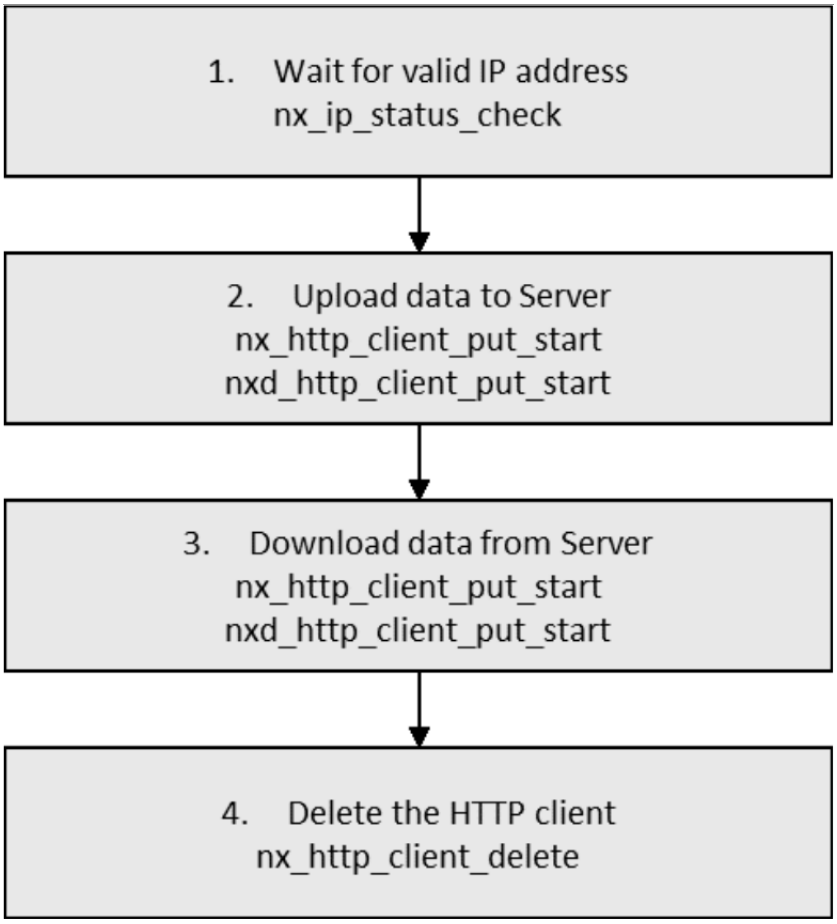


Figure 459: Flow Diagram of a Typical NetX/NetX Duo HTTP Client Module Application

4.3.22 NetX/NetX Duo HTTP Server

4.3.22.1 NetX/NetX Duo HTTP Server Introduction

The Hypertext Transfer Protocol (HTTP) utilizes reliable Transmission Control Protocol (TCP) services to perform its content transfer function; all operations on the Web utilize the HTTP protocol. The NetXTM Duo HTTP Server accommodates both IPv4 and IPv6 networks, while the NetXTM HTTP Server only supports IPv4 communications. IPv6 does not directly affect the HTTP protocol; however, some differences with the NetX HTTP Server are necessary to accommodate IPv6 and are noted in this document.

Note

Except for internal processing, the NetX Duo HTTP Server module is almost identical in the application, set up and running of an HTTP session as the NetX HTTP Server module. Where they differ is noted in this document.

Unsupported Features

HTTP Insert GMT Date Header Callback has not been tested in this version of SSP. HTTP Cache Info Get Callback has not been tested in this version of SSP.

NetX/NetX Duo HTTP Server Module Features

- Compliant with Request for Comments (RFC) RFC1945 Hypertext Transfer Protocol/1.0, RFC 2581 TCP Congestion Control, RFC 1122 Requirements for Internet Hosts, and related RFCs.
- Multipart support
- Basic and digest authentication support
- Callback support for several key functions:
    - HTTP Authentication Callback
    - HTTP Request Notify Callback
    - HTTP Invalid Username/Password Callback
    - HTTP Insert GMT Date Header Callback
    - HTTP Cache Info Get Callback

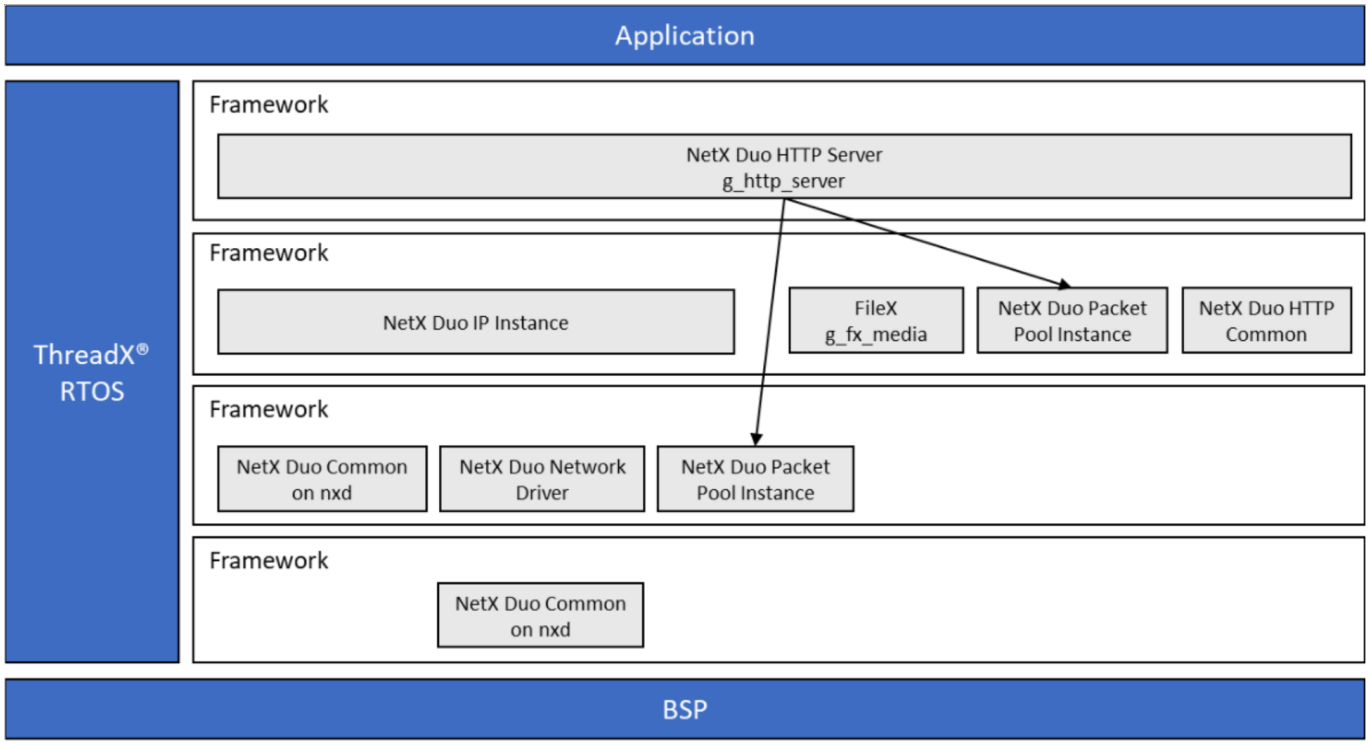


Figure 460: NetX/NetX Duo HTTP Server Module Block Diagram

Note

In the figure above, the FileX and NetX (or NetX Duo) Network Driver modules have multiple implementation options available. See the descriptions just after the module stack figure in Including the NetX/NetX Duo HTTP Server Module in an Application for additional details.

4.3.22.2 NetX/NetX Duo HTTP Server Module APIs Overview

The NetX HTTP Server module defines APIs for creating, deleting, generating response packets, response sending and getting information from a received packet. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo HTTP Server Module API Summary

Function Name	Example API Call and Description
nx_http_server_cache_info_callback_set	nx_http_server_cache_info_callback_set(&my_server, cache_info_get); Set callback to retrieve age and last modified date of specified URL.

nx_http_server_callback_data_send	nx_http_server_callback_data_send(server_ptr, "HTTP/1.0 200 rnContent-Length: 103rnContent-Type: text/htmlrnrn",63);
	nx_http_server_callback_data_send(server_ptr, "<HTML>rn<HEAD><TITLE>NetX HTTP Test </TITLE></HEAD>rn <BODY>rn<H1>NetX Test Page </H1>rn</BODY>rn</HTML>rn", 103);
	Send HTTP data from callback function.

nx_http_server_callback_generate_response_header	nx_http_server_callback_generate_response_header (server_ptr, &packet_ptr, status_code, content_length, content_type, additional_header); Create response header in callback functions.
nx_http_server_callback_packet_send	nx_http_server_callback_packet_send(server_ptr, packet_ptr); Send an HTTP packet from an HTTP callback.
nx_http_server_callback_response_send	nx_http_server_callback_response_send(server_ptr,"HTTP/1.0 404 ", "NetX HTTP Server unable to find file: ", resource); Send response from callback function.
nx_http_server_content_get	nx_http_server_content_get(server_ptr, packet_ptr, 0, my_buffer, 100, &actual_size); Get content from the request.
nx_http_server_content_get_extended	nx_http_server_content_get_extended(server_ptr, packet_ptr, 0, my_buffer, 100, &actual_size); Get content from the request; supports empty (zero Content Length) requests.
nx_http_server_content_length_get	nx_http_server_content_length_get(packet_ptr); Get length of content in the request. Length is the status return value. A length of zero indicates an error.
nx_http_server_content_length_get_extended	nx_http_server_content_length_get_extended(packet_ptr, &content_length); Get length of content in the request; supports empty (zero Content Length) requests.

nx_http_server_create	nx_http_server_create(&my_server, "my server", &ip_0, &ram_disk, stack_ptr, stack_size, &pool_0, my_authentication_check, my_request_notify); Create an HTTP Server instance.
nx_http_server_delete	nx_http_server_delete(&my_server); Delete an HTTP Server instance.
nx_http_server_get_entity_content	nx_http_server_get_entity_content(server_ptr, &packet_ptr, &offset, &length); Return size and location of entity content in URL.
nx_http_server_get_entity_header	nx_http_server_get_entity_header(server_ptr, &packet_ptr, entity_header_buffer, buffer_size); Extract URL entity header into specified buffer.
nx_http_server_gmt_callback_set	nx_http_server_gmt_callback_set(&my_server, gmt_get); Set callback to retrieve GMT date and time.
**nx_http_server_invalid_userpassword_notify_set	nx_http_server_invalid_userpassword_notify_set(&my_server, invalid_username_password_callback); Set callback for when invalid username and password is received in a Client request.
nx_http_server_mime_maps_additional_set	nx_http_server_mime_maps_additional_set(&my_server, &my_mime_maps[0], 2); Define additional mime maps for HTML.
nx_http_server_packet_content_find	nx_http_server_packet_content_find(server_ptr, packet_ptr, &content_length); Extract content length in HTTP header and set pointer to start of content data.
nx_http_server_packet_get	nx_http_server_packet_get(server_ptr, &packet_ptr); Receive client packet directly.
nx_http_server_param_get	nx_http_server_param_get(packet_ptr, 0, param_destination, 30); Get parameter from the request.
nx_http_server_query_get	nx_http_server_query_get(packet_ptr, 0, query_destination, 30); Get query from the request.
nx_http_server_start	nx_http_server_start(&my_server); Start the HTTP Server.
nx_http_server_stop	nx_http_server_stop(&my_server); Stop the HTTP Server.
nx_http_server_type_get	nx_http_server_type_get(server_ptr, resource_name, type_string); Extract HTTP type e.g. text/plain from header. Type is returned in the status return. A value of zero indicates an error.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo HTTP Server. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	Successfully performed function
NX_PTR_ERROR**	Invalid pointer input
NX_CALLER_ERROR **	Invalid caller of service
NX_HTTP_DATA_END	End of request content
NX_HTTP_TIMEOUT	HTTP Server timeout in getting next packet of content
NX_CALLER_ERROR	Invalid caller of this service
NX_HTTP_INCOMPLETE_PUT_ERROR	Improper HTTP header format
NX_HTTP_POOL_ERROR	Packet payload of pool is not large enough to contain complete HTTP request
NX_HTTP_BOUNDARY_ALREADY_FOUND	Content for the HTTP server internal multipart markers is already found
NX_HTTP_NOT_FOUND	Entity header field or client request parameter or multipart component not found
NX_HTTP_IMPROPERLY_TERMINATED_PARAM	Client request parameter not properly terminated
NX_HTTP_NO_QUERY_PARSED	Server unable to find query in client request
NX_HTTP_TIMEOUT	No packet received in the specified wait interval
NX_HTTP_ERROR	Internal HTTP Server error
NX_HTTP_SERVER_DEFAULT_MIME	No matching extension type found. Return the default MIME type. Not an error status.
NX_SIZE_ERROR	Invalid total size of resource in PUT request
NX_INVALID_PACKET	Invalid TCP packet; not enough room for packet header

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

**These are error codes which are only returned if error checking is enabled. Refer to the *NetX User's Guide* for the Renesas Synergy Platform or *NetX Duo User's Guide* for the Renesas Synergy Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.22.3 NetX/NetX Duo HTTP Server Module Operational Overview

The NetX HTTP Server module creates an IP instance that carries out NetX operations and enables it for TCP services in the NetX library; it then creates the HTTP Server instance and TCP socket for listening to client requests on port 80. The HTTP Server requires a packet pool; the module can supply one either by sharing the IP default packet pool (g_packet_pool0) or create a new one. The minimum packet payload is set by the Minimum size of packets in pool property of the HTTP Server module. This packet pool is used by the HTTP Server only to transmit packets, so the packet pool size and payload can be optimized on the expected size and number of HTTP Server packets sent out.

The NetX Duo HTTP Server supports both IPv4 and IPv6 connections. If the HTTP Server has clients desiring to connect over IPv6, make sure the NetX Duo IPv6 Support property is enabled in the NetX Duo Source element. It may be necessary to enable ICMPv6 checksum computation for the underlying ICMPv6 protocols. To do so, set the Checksum computation support on transmitted ICMPv6 packets and Checksum computation support on received ICMPv6 packets properties of the NetX Duo Source element to Enabled. (If the host hardware automatically computes ICMPv6 checksums, these can be left disabled.) Make sure the IPv6 Global Address of the Client host is set in the IP instance element. Thereafter, the NetX Duo does the necessary processing to enable IPv6 and ICMPv6 services required for IPv6 underlying protocols.

The HTTP Server is also designed for use with the FileX® embedded file system.

HTTP Server Responses

When the HTTP Server processes the client command, it returns an ASCII response string that includes a 3-digit numeric status code. The numeric response is used by the HTTP Client software to determine whether the operation succeeded or failed. Following is a list of various HTTP Server responses to client commands:

HTTP Server responses to client commands

Numeric Field	Meaning
200	Request was successful
400	Request was not formed properly
401	Unauthorized request, client needs to send authentication
404	Specified resource in request was not found
500	Internal HTTP Server error
501	Request not implemented by HTTP Server
502	Service is not available

For example, a successful client request to PUT the file test.htm is responded to with the message HTTP/1.0 200 OK.

HTTP Authentication

HTTP authentication is optional and is not required for all web requests. There are two types of authentication, basic and digest. Basic authentication is equivalent to the name and password authentication found in many protocols. In HTTP basic authentication, the name and passwords are concatenated and encoded in the base64 format. The main disadvantage of basic authentication is the name and password are transmitted openly in the request, making it easy for the name and

password to be stolen. Digest authentication addresses this problem by never transmitting the name and password in the request. Instead, an algorithm is used to derive a 128-bit key or digest from the name, password, and other information. The NetX HTTP Server supports the standard MD5 digest algorithm.

The HTTP Server authentication callback can decide if a requested resource requires authentication. If authentication is required and the client request did not include the proper authentication, an HTTP/1.0 401 Unauthorized response with the type of authentication required is sent to the client. The client is then expected to form a new request with the proper authentication.

HTTP Authentication Callback

The HTTP Server authentication callback routine is specified by the Name of Authentication Checking Function property of the HTTP Server Thread. This function is called at the beginning of each HTTP Client request.

The callback routine provides the NetX HTTP Server with the username, password, and realm strings associated with the resource and returns the type of authentication necessary. If no authentication is necessary for the resource, the authentication callback should return the value of NX_HTTP_DONT_AUTHENTICATE. If basic authentication is required for the specified resource, the routine should return NX_HTTP_BASIC_AUTHENTICATE. If MD5 digest authentication is required, the callback routine should return NX_HTTP_DIGEST_AUTHENTICATE.

The format of the authenticate callback routine is defined as:

```
UINT nx_http_server_authentication_check(NX_HTTP_SERVER *server_ptr, UINT
request_type, CHAR *resource, CHAR **name, CHAR **password, CHAR **realm);
```

The input parameters are defined as follows:

Input Parameters Definitions

Parameter	Meaning
request_type	Specifies the HTTP Client request, valid requests are defined as: NX_HTTP_SERVER_GET_REQUEST NX_HTTP_SERVER_POST_REQUEST NX_HTTP_SERVER_HEAD_REQUEST NX_HTTP_SERVER_PUT_REQUEST NX_HTTP_SERVER_DELETE_REQUEST
resource	Specific resource requested.
name	Destination for the pointer to the required username.
password	Destination for the pointer to the required password.
realm	Destination for the pointer to the realm for this authentication.

Name, password, and realm pointers are not used if NX_HTTP_DONT_AUTHENTICATE is returned by the authentication callback routine. The HTTP Server developer must ensure that the maximum size of the username and password (defined by the Maximum username length and Maximum password length properties of the NetX HTTP Common) are large enough for the username and password specified in the authentication callback. These are both defaulted to size 20 characters.

HTTP Server Request Notify callback

If a request callback is specified, (the Name of Request Notify Callback Function property of the NetX HTTP Server module) the NetX HTTP Server forwards requests to the specified function after authentication and validity of the client request is completed without errors. The callback should indicate (by the return status) if no more processing of the client request is required (return status NX_HTTP_CALLBACK_COMPLETED), if there was an error in the callback processing, (status is non-zero), or the process was completed successfully and the HTTP Server should continue processing the client request. The format of this callback is:

```
UINT        request_notify(NX_HTTP_SERVER *server_ptr, UINT request_type, CHAR
*resource,
NX_PACKET *packet_ptr)
```

To disable the request notify callback, set the Name of Request Notify Callback Function property to NULL.

HTTP Invalid Username/Password Callback

The optional Invalid Username/Password callback in the NetX HTTP Server module is invoked if the HTTP Server receives an invalid username-and-password combination in a client request. To set the Invalid Username/Password callback function, use the nx_http_server_invalid_user_password_set service. Note that for NetX Duo HTTP Server module, this takes a NXD_ADDRESS *client_ip_address, while NetX HTTP Server module takes a ULONG client_ip_address.

HTTP Insert GMT Date Header Callback

The NetX HTTP Server supports an optional callback to insert a date header in its response messages. This callback is invoked when the Server responds to a Client PUT or GET request. To set the GMT callback use the nx_http_server_gmt_callback service before starting the NetX HTTP Server thread.

HTTP Cache Info Get Callback

The NetX HTTP Server has an optional callback to request the maximum age and date from the HTTP application for a specific resource. This information is used to determine if the HTTP server sends the entire page in response to a Client Get request. If the if modified since in the Client request is not found or does not match the last modified date returned by the get-cache callback, the entire page is sent. To set a cache callback function, use the nx_http_server_cache_info_callback_set service.

HTTP Multipart Support

Multipurpose Internet Mail Extensions (MIME) was originally intended for the SMTP protocol, but its use has spread to HTTP. MIME allows messages to contain mixed message types (for example, image/jpg and text/plain) within the same message. The NetX HTTP Server has added services to

determine content type in HTTP messages containing MIME from the client. To enable multipart support, set the Multipart HTTP requests support property of the NetX HTTP Server module to enable.

NetX/NetX Duo HTTP Server Module Important Operational Notes and Limitations

NetX/NetX Duo HTTP Server Module Operational Notes

- The NetX HTTP Server module requires a FileX media (Block media or USB Mass Storage). When an HTTP Server stack element is added to the project, an Add FileX box is attached to it. The configurator automatically sets up and initializes the FileX media for the server before the server is started. For more details for configuring FileX, see *FileX™ User's Guide for the Renesas Synergy™ Platform*.
- The NetX HTTP Server also requires a packet pool for transmitting packets. It can share the IP default packet pool or create a separate packet pool. See the section on *Including the NetX HTTP Server Module in an Application* for details on setting the HTTP Server packet pool.

NetX/NetX Duo HTTP Server Module Limitations

- Persistent connections are not supported.
- Request pipelining is not supported.
- The HTTP Server supports both basic and MD5 digest authentication, but not MD5-sess.
- No content compression is supported.
- Trace, Options, and Connect requests are not supported.
- The packet pool associated with the HTTP Server must be large enough to hold the complete HTTP header.

4.3.22.4 Including the NetX/NetX Duo HTTP Server Module in an Application

This section describes how to include either or both the NetX and NetX Duo HTTP Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo HTTP Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo HTTP Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_http_server0 NetX HTTP Server	Threads	New Stack> X-Ware> NetX> Protocols> NetX HTTP Server
g_http_server0 NetX Duo HTTP Server	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo HTTP Server

When the NetX and/or NetX Duo HTTP Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

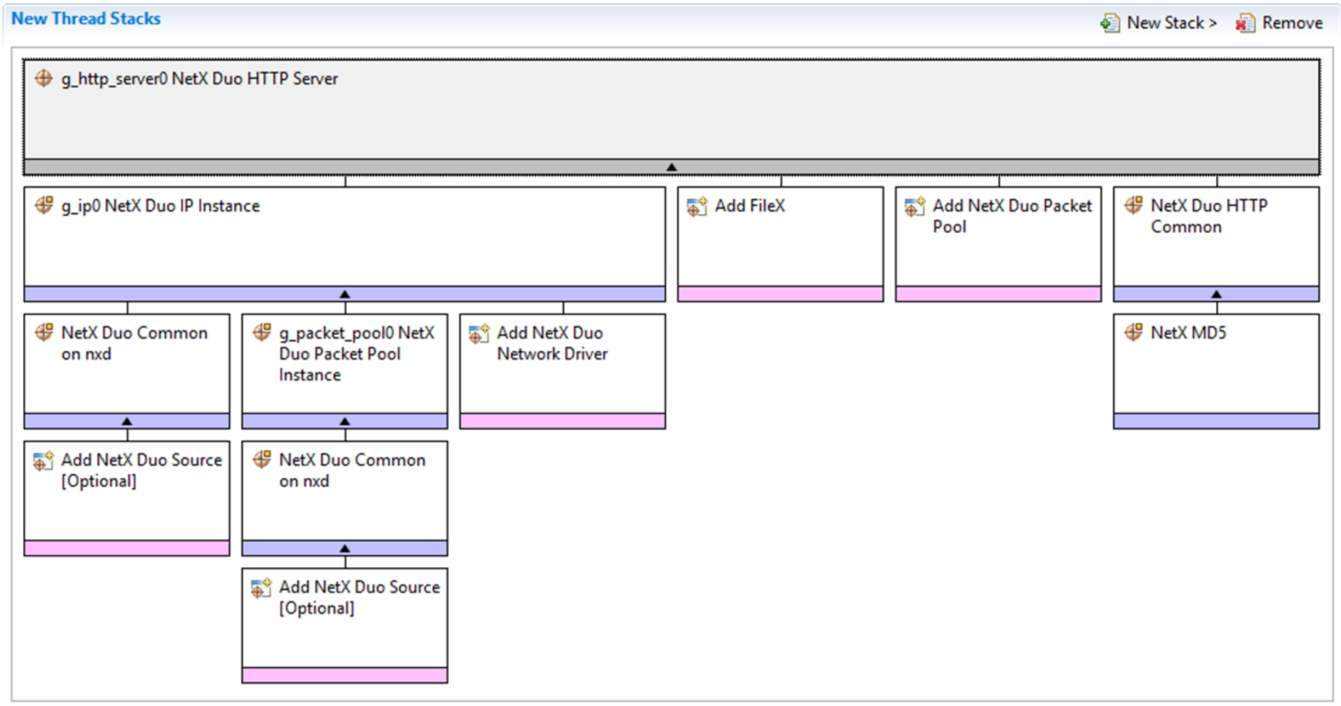


Figure 461: NetX/NetX Duo HTTP Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

Additionally, in the stack above, the FileX stack has also not been populated yet. There are multiple possible selections for the FileX module; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- FileX Stub
- FileX on Block Media (implemented on Block Media Framework on sf_block_media_ram)
- FileX on USB Mass Storage (implemented on USBX Host Class Mass Storage)

4.3.22.5 Configuring the NetX/NetX Duo HTTP Server Module

The NetX/NetX Duo HTTP Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo HTTP Server Module

ISDE Property	Value	Description
FileX Support	Enable, Disable Default: Enable	FileX support selection
Multipart HTTP requests support	Enable, Disable Default: Disable	Multipart HTTP requests support selection
Internal thread priority	16	Internal thread priority selection
Internal thread time slicing interval (bytes)	NetX Default: 1 NetX Duo Default: 2	Internal thread time slicing interval selection
Server socket window size (bytes)	2048	Server socket window size selection
Server time out (seconds)	10	Server time out selection
Server time out for accept (seconds)	10	Server time out for accept selection
Server time out for disconnect (seconds)	10	Server time out for disconnect selection
Server time out for receive (seconds)	10	Server time out for receive selection
Server time out for send (seconds)	10	Server time out for send selection
Maximum size of header field (bytes)	256	Maximum size of header field selection
Maximum connections in queue	5	Maximum connections in queue selection

Maximum client user user name length (bytes)	20	Maximum client user password length selection
Maximum client user password length (bytes)	20	Minimum size of packets in pool selection
Minimum size of packets in pool (bytes)	600	Minimum size of packets in pool selection
Name	g_http_server0	Module name
Internal thread stack size (bytes)	4096	Internal thread stack size selection
Name of Authentication Checking Function	authentication_check	Name of Authentication Checking Function selection
Name of Request Notify Callback Function	request_notify	Name of Authentication Checking Function selection
Name of generated initialization function	http_server_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different IP masks and addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo HTTP Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection

Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo HTTP Common Instance

ISDE Property	Value	Description
Type of Service	Normal, Minimum delay, Maximum data, Maximum reliabililty, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection
Time to live	128	Time to live selection
MD5 Support	Enable, Disable Default: Disable	MD5 support selection
Maximum resource name length (bytes)	40	Packet queue depth selection
**Maximum password length (bytes)	20	Maximum password length selection
**Maximum username length (bytes)	20	Minimum username length selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection

Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo HTTP Server Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo HTTP Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description

Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.22.6 Using the NetX/NetX Duo HTTP Server Module in an Application

The steps in using the NetX/NetX Duo HTTP Server module in a typical application are:

Auto Generated code to initialize NetX and NetX Duo HTTP Server in the Application (common_data.c)

- Create HTTP Packet pool using the nx_packet_pool_create API
- Create IP Instance using the nx_ip_create API
- Enable ARP using the nx_arp_enable API
- Enable TCP using the nx_tcp_enable API
- Create HTTP Server using the nx_http_server_create API

User Application Code (<thread>_entry.c)

1. Wait for valid IP address using the nx_ip_status_check API.
2. Start HTTP Server using the nx_http_server_start API.
3. Handle optional callbacks if registered with the HTTP Server (Authentication Check, Request Notify, GMT set, Cache get and Invalid Username).
4. Stop HTTP Server using the nx_http_server_stop API.
5. Delete HTTP Server using the nx_http_server_delete API.

Note

If the server packet pool is used only by the server, this can be deleted too using the nx_packet_pool_delete API.

Users do not have to worry about auto-generated code. Auto-generated code is included once the

user generates the project after configuring the stack. Users only need to write the user application code in the associated file (typically <thread>_entry.c).

The following figure illustrates common steps in a typical operational flow diagram:

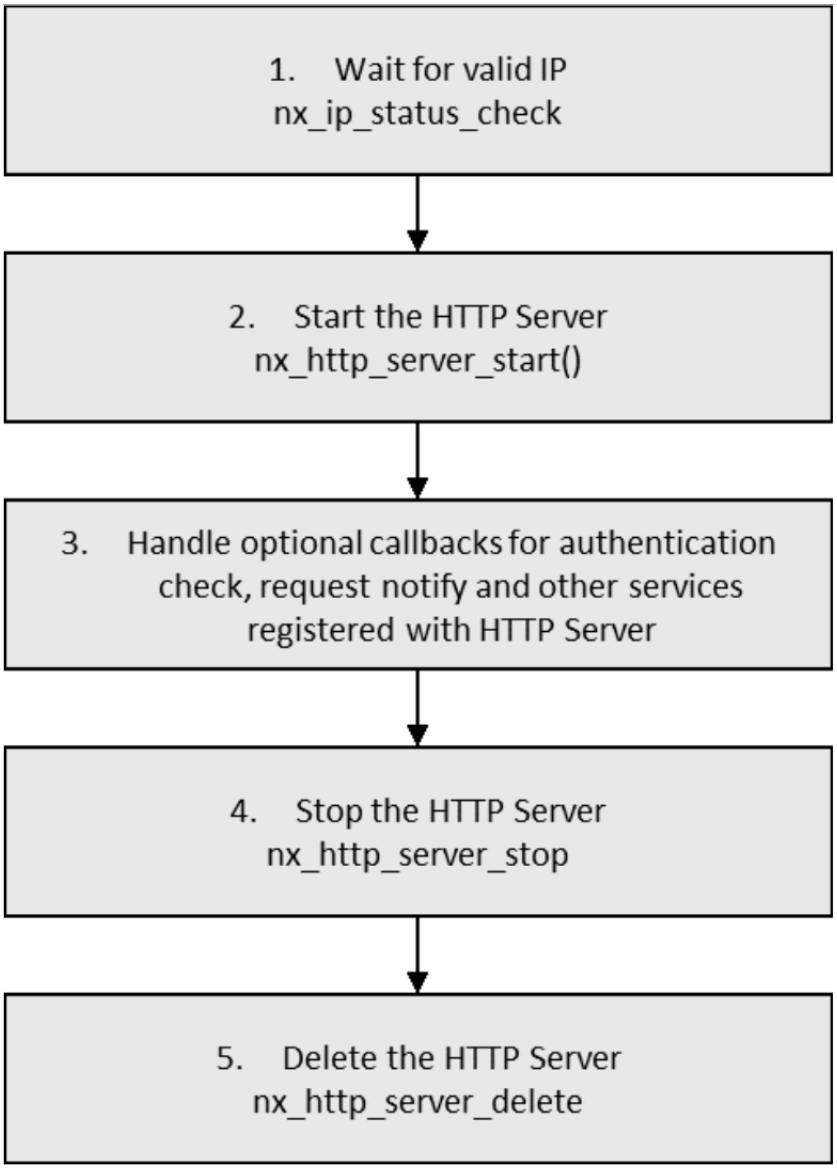


Figure 462: Flow Diagram of a Typical NetX/NetX Duo HTTP Server Module Application

4.3.23 NetX Duo HTTP Client (HTTPS/HTTPS 1.1)

4.3.23.1 NetX Duo Web HTTP/HTTPs Client Introduction

The NetX Web HTTP/HTTPs Client module provides a high-level API for Hyper Text Transport Protocol(HTTP) for transferring content on the web. The HTTP protocol utilizes Transmission Control Protocol (TCP) services to perform its content transfer function. HTTPs is the secure version of the

HTTP protocol which uses HTTP on top of the Transport Layer Security (TLS) protocol to secure underlying TCP connection.

HTTP/HTTPs client is implemented on top of NetX Duo IP and NetX Duo Packet Pool. NetX Duo IP attaches itself to appropriate link layer driver such ethernet/Wi-Fi/cellular. HTTP client can optionally connect to HTTP server over secure connection and in such case, it uses service provided by NetX Duo TLS Common

Unsupported Features

- HTTP Client authentication using username and password integrated but not tested. Request pipelining is not supported
- At present, the HTTP Client supports only basic authentication. When using TLS for HTTPS, HTTP authentication may still be used.
- No content compression is supported.
- TRACE, OPTIONS, and CONNECT requests are not supported.
- The packet pool associated with the HTTP Server or Client must be large enough to hold the complete HTTP header.

NetX Duo Web HTTP/HTTPs Client Module Features

- NetX Web HTTP/HTTPs client is compliant with below RFCs
    - RFC1945 "Hypertext Transfer Protocol/1.0"
    - RFC 2616 "Hypertext Transfer Protocol – HTTP/1.1"
    - RFC 2818 "HTTP Over TLS"
    - RFC 2581 "TCP Congestion Control"
    - RFC 1122 "Requirements for Internet Hosts", and related RFCs.
    - RFC 2818 "HTTP over TLS" for HTTPS
- Support HTTP Get/POST/HEAD/PUT/DELETE commands. Note that these commands are generated internally by NetX Web HTTP/HTTPs client APIs.
- Support basic authentication. When using TLS for HTTPs, HTTP authentication may still be used
- Provides option to enable/disable TLS for secure communication using NetX Secure in SSP

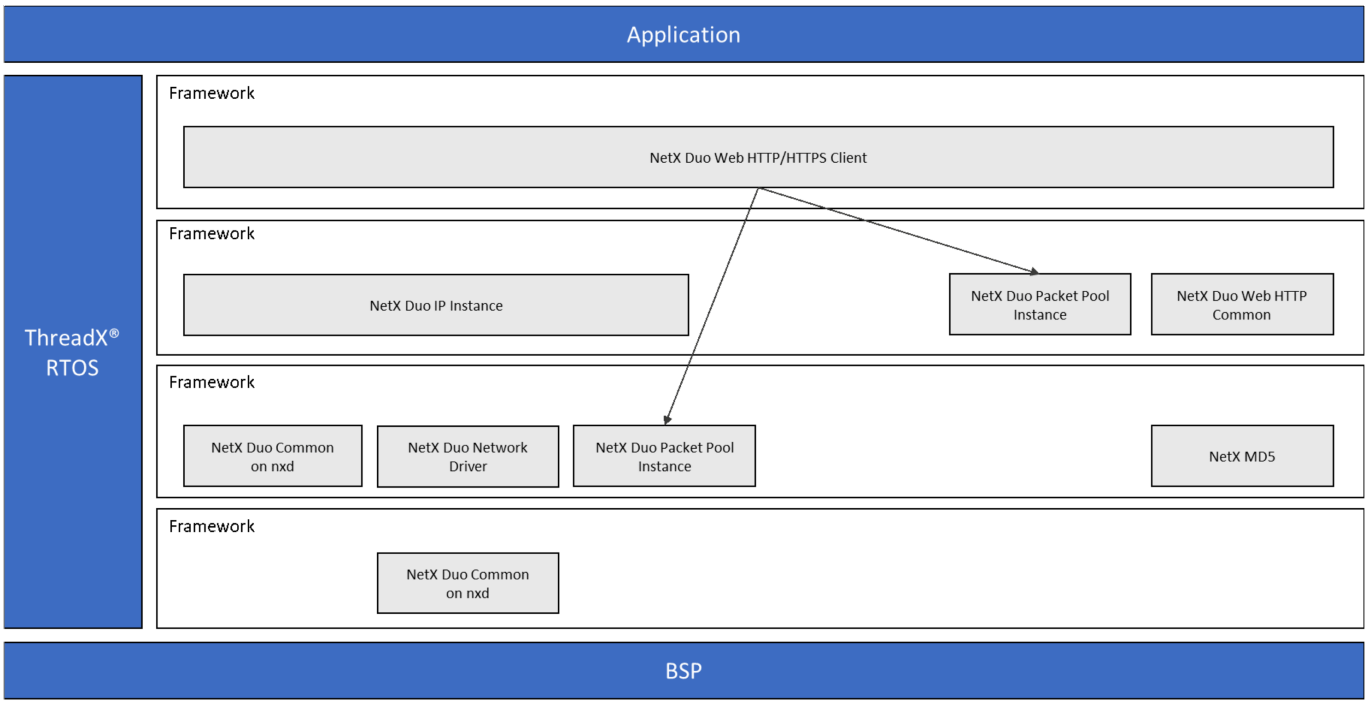


Figure 463: NetX Duo Web HTTP/HTTPs Client Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo Web HTTP/HTTPs Client Module in an Application for additional details.

4.3.23.2 NetX Duo Web HTTP/HTTPs Client Module APIs Overview

The NetX Duo Web HTTP/HTTPs Client module defines APIs for creating, deleting, getting and putting. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo Web HTTP/HTTPs Client Module API Summary

Function Name	Example API Call and Description
nx_web_http_client_connect	nx_web_http_client_connect(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS *server_ip, UINT server_port, ULONG wait_option); Open a plaintext socket to an HTTP server for custom requests.
nx_web_http_client_create	nx_web_http_client_create(NX_WEB_HTTP_CLIENT *client_ptr, CHAR *client_name, NX_IP *ip_ptr, NX_PACKET_POOL *pool_ptr, ULONG window_size); Create an HTTP Client Instance.

nx_web_http_client_delete	nx_web_http_client_delete(NX_WEB_HTTP_CLIENT *client_ptr); Delete an HTTP Client Instance.
nx_web_http_client_delete_start	nx_web_http_client_delete_start(NX_WEB_HTTP_CLIE NT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG wait_option); Start a plaintext HTTP DELETE request.
nx_web_http_client_delete_secure_start	nx_web_http_client_delete_secure_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session), ULONG wait_option); Start an encrypted HTTPS DELETE request.
nx_web_http_client_get_start	nx_web_http_client_get_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG wait_option); Start a plaintext HTTP GET request.
nx_web_http_client_get_secure_start	nx_web_http_client_get_secure_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session), ULONG wait_option); Start an encrypted HTTPS GET request.
nx_web_http_client_head_start	nx_web_http_client_head_start(NX_WEB_HTTP_CLIEN T *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG wait_option); Start a plaintext HTTP HEAD request.

nx_web_http_client_head_secure_start	nx_web_http_client_head_secure_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session), ULONG wait_option); Start an encrypted HTTPS HEAD request.
nx_web_http_client_post_start	nx_web_http_client_post_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG total_bytes, ULONG wait_option); Start an HTTP POST request.
nx_web_http_client_post_secure_start	nx_web_http_client_post_secure_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG total_bytes, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session), ULONG wait_option); Start an encrypted HTTPS POST request.
nx_web_http_client_put_start	nx_web_http_client_put_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG total_bytes, ULONG wait_option); Start an HTTP PUT request.
nx_web_http_client_put_secure_start	nx_web_http_client_put_secure_start(NX_WEB_HTTP_CLIENT *client_ptr, NXD_ADDRESS ip_address, UINT server_port, CHAR *resource, CHAR *host, CHAR *username, CHAR *password, ULONG total_bytes, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session), ULONG wait_option); Start an encrypted HTTPS PUT request.

nx_web_http_client_put_packet	nx_web_http_client_put_packet(NX_WEB_HTTP_CLIEN T *client_ptr, NX_PACKET *packet_ptr, ULONG wait_option); Send next resource data packet.
nx_web_http_client_request_header_add	nx_web_http_client_request_header_add(NX_WEB_HTTP_CLIENT *client_ptr, CHAR *field_name, UINT name_length, CHAR *field_value, UINT value_length, UINT wait_option); Add a custom header to a custom HTTP request.
nx_web_http_client_request_initialize	nx_web_http_client_request_initialize (NX_WEB_HTTP_CLIENT *client_ptr, UINT method, CHAR *resource, CHAR *host, UINT input_size, UINT transfer_encoding_trunked, CHAR *username, CHAR *password, UINT wait_option); Initialize a custom HTTP request.
nx_web_http_client_request_send	nx_web_http_client_request_send(NX_WEB_HTTP _CLI ENT *client_ptr, UINT wait_option); Send a custom HTTP request.
nx_web_http_client_response_body_get	nx_web_http_client_response_body_get(NX_WEB_HTTP_CLIENT *client_ptr, NX_PACKET **packet_ptr, ULONG wait_option); Get next resource data packet.
nx_web_http_client_response_header_callback_s et	nx_web_http_client_response_header_callback_s et(NX_WEB_HTTP_CLIENT *client_ptr, VOID (*callback_function)(NX_WEB_HTTP_CLIENT *client_ptr, CHAR *field_name, UINT field_name_length, CHAR *field_value, UINT field_value_length)); Set callback to invoke when processing HTTP headers.
nx_web_http_client_secure_connect	nx_web_http_client_secure_connect(NX_WEB_HT TP_C LIENT *client_ptr, NXD_ADDRESS *server_ip, UINT server_port, UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls), ULONG wait_option); Open a TLS session to an HTTPS server for custom requests.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures

and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful connection of TCP socket.
NX_PTR_ERROR	Invalid pointer input.
NX_WEB_HTTP_NOT_READY	Another request is already in progress.
NX_WEB_HTTP_POOL_ERROR	Invalid payload size in packet pool
NX_CALLER_ERROR	Invalid caller of this service.
NX_HTTP_PASSWORD_TOO_LONG	Password exceeded expected length
NX_WEB_HTTP_ERROR	Internal HTTP Client error.
NX_WEB_HTTP_NOT_READY	HTTP Client not ready.
NX_WEB_HTTP_FAILED	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	Invalid name and/or password.
NX_WEB_HTTP_USERNAME_TOO_LONG	Username too large for buffer.
NX_SIZE_ERROR	Invalid total size of resource.
NX_WEB_HTTP_REQUEST_UNSUCCESSFUL_CODE	Received Server error code
NX_WEB_HTTP_BAD_PACKET_LENGTH	Invalid packet length.
NX_WEB_HTTP_INCOMPLETE_PUT_ERROR	Server responds before PUT is complete.
NX_INVALID_PACKET	Packet too small for TCP header.
NX_WEB_HTTP_METHOD_ERROR	Some required information was missing (e.g. input_size for PUT or POST).
NX_WEB_HTTP_GET_DONE	HTTP Client get packet is done.

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.23.3 NetX Duo Web HTTP/HTTPs Client Module Operational Overview

HTTP (Hyper Text Transport Protocol) is used to exchange hypertext between HTTP client and server. An HTTP client initiates a HTTP request such as Get/POST/HEAD/PUT/DELETE by establishing a TCP connection to particular port on HTTP server e.g. port 80, port 443. An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status message such as "HTTP/1.1 200 ok" followed by a message of its own.

The NetX Web HTTP/HTTPs client module can be used in the normal mode or secure mode

NetX Web HTTP/HTTPs client module Normal Mode Operational Description

In normal mode, the communication between HTTP client and server is not secure.

Netx Web HTTP/HTTPs client module Secure Mode Operational Description

In Secure mode, the communication between HTTP client and server is secured using the TLS protocol. In the thread pane, TLS protocol is represented by "Add NetX Duo TLS common [Optional]" block as shown in the figure below

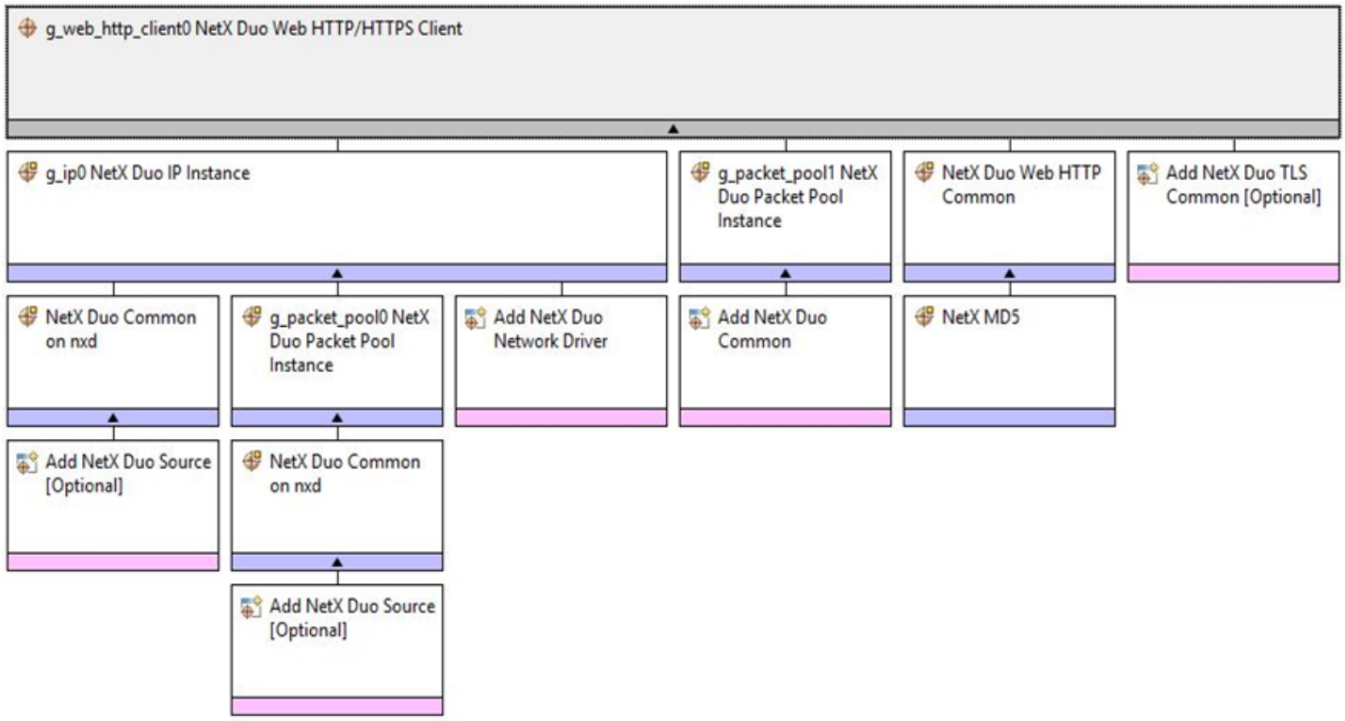


Figure 464: NetX Duo Web HTTP/HTTPs Client Module Component Thread-Pane View

Adding NetX Duo TLS Common block enables TLS support and internally defines the NX_SECURE_ENABLE macro. The figure below shows the thread pane view of HTTP client with TLS support enabled.

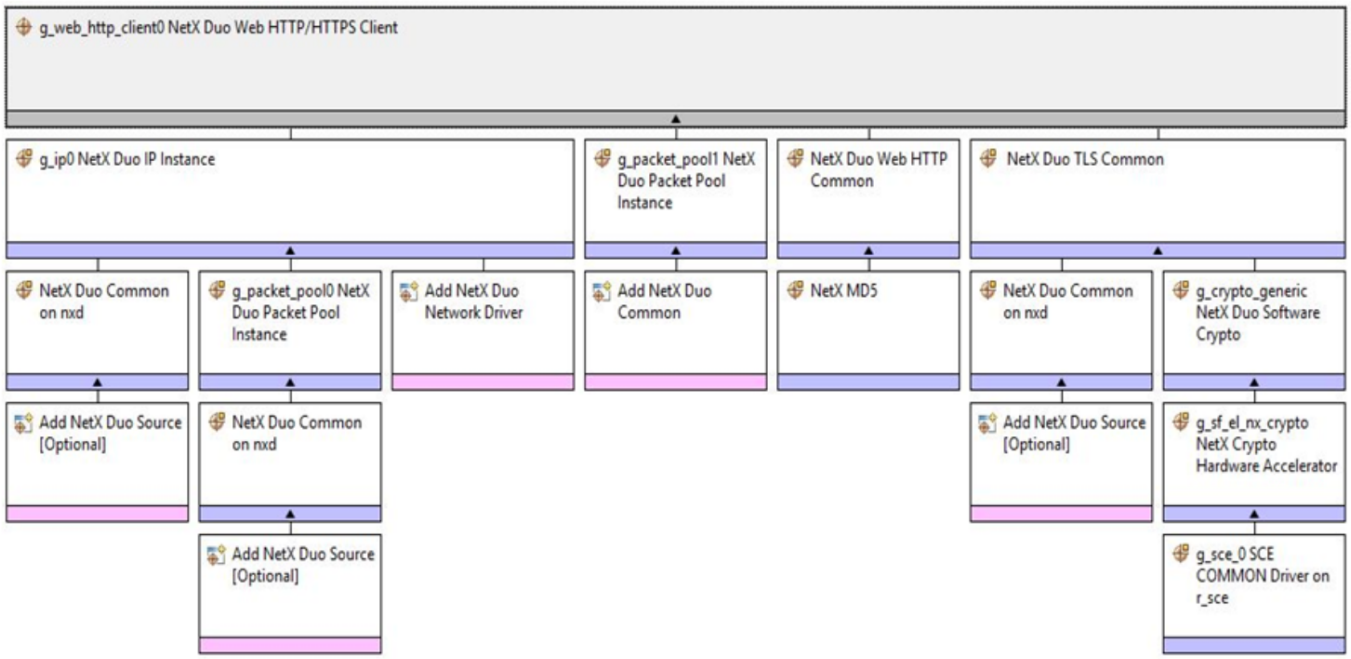


Figure 465: NetX Duo Web HTTP/HTTPs Client Module Component with TLS Support Enabled

NetX Duo Web HTTP/HTTPs Client Module Important Operational Notes and Limitations

NetX Duo Web HTTP/HTTPs Client Module Operational Notes

The NetX Web HTTP/HTTPs Client component is added by clicking on the (+) sign in the thread pane window -> Azure RTOS -> NetX Duo -> Protocols -> NetX Web HTTP/HTTPs Client.

Adding the NetX Web HTTP/HTTPs Client component to a project automatically adds the option to add the NetX Duo TLS component required for secure HTTP client.

The NetX Web HTTP/HTTPs Client properties are listed in the following table:

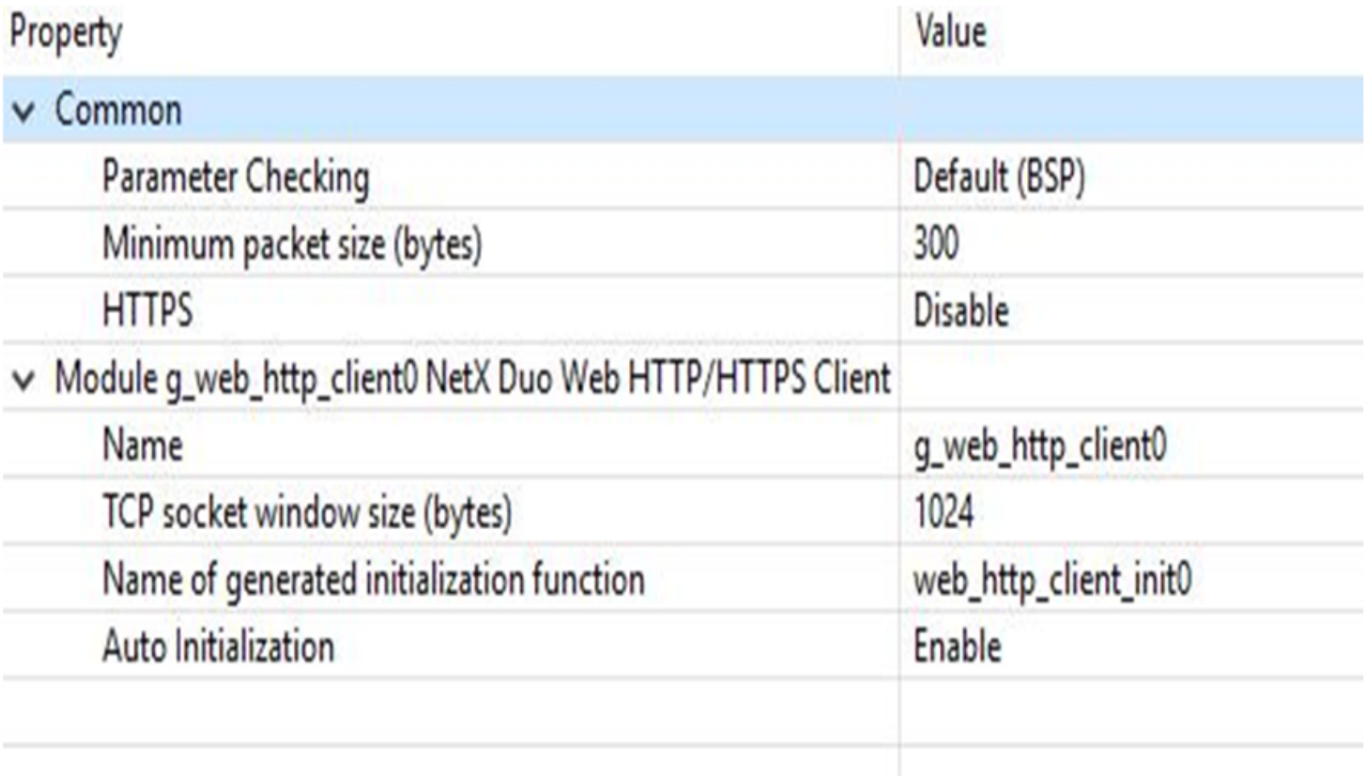


Figure 466: NetX Duo Web HTTP/HTTPs Client Module Component Configurable Properties

In the figure above, "Common" properties are those configurable options in the NetX Web HTTP/HTTPs Client that are common to all instances of the HTTP/HTTPs client in the project. The "Module" properties are specific to each instance of HTTP/HTTPs Client in the project.

Common Properties

- Parameter Checking: This enables/disables basic HTTP error checking. It is typically used after the application has been debugged. The default value is BSP.
- Minimum Packet Size (bytes): The minimum size of the packets in the pool specified at Client creation. The minimum size is needed to ensure the complete HTTP header can be contained in one packet. The default value is 300 bytes.
- HTTPS: This enables/disables TLS which is used to establish secure channel. The default value is disable.

Module Properties

- Name: Name of the HTTP/HTTPs client instance.
- TCP socket window size(bytes) : Set the size of client's TCP socket receive window size. The default is 1024 bytes.
- Name of generated initialization function: Name of initialization function which creates HTTP/HTTPs client instance. The default is the auto-generated function named web_http_client_init0.
- Auto Initialization: Enable/disable call to initialization function. This determines if the function specified in the Name of Generated Initialization function option is called. If set to Enable, it will invoke this function. Otherwise if set to Disable, the application must call the nx_web_http_client_create() API before using any NetX Web HTTP/HTTPs Client services.

NetX Duo Web HTTP Common Configurable Properties

The common properties are listed in the following table:

Property	Value
Common	
Type of Service	Normal
Fragmentation option	Don't fragment
MD5 Support	Disable
Time to live	128
Maximum password length (bytes)	20
Maximum username length (bytes)	20

Figure 467: NetX Duo Web HTTP/HTTPs Client Module Component Common Configurable Properties

- Type of Service: Type of network service required for the HTTP TCP requests. The network service can be of type (Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost) and the default value is Normal.
- Fragmentation Option: This enables/disables TCP fragmentation for HTTP client request. The default value is Don't Fragment.
- MD5 Support: This enables/disables MD5 digest support required for digest authentication. The default value is disable.
- Time to live: This specifies maximum no. of routers the HTTP packet can pass through before it get discarded. The default is 128.
- Maximum password length (bytes): This defines the maximum length of client supplied password. The default is 20.
- The Maximum username length (bytes): This defines the maximum length of client supplied username. The default is 20.

NetX Duo Web HTTP/HTTPs Client Module Limitations

- Request pipelining is not supported
- No content compression is not supported
- TRACE, OPTIONS, and CONNECT HTTP requests are not supported
- The packet pool associated with client must be large enough to hold the complete HTTP header
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.23.4 Including the NetX Duo Web HTTP/HTTPs Client Module in an Application

This section describes how to include either or both the NetX Duo Web HTTP/HTTPs Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo Web HTTP/HTTPs Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo Web HTTP/HTTPs Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_web_http_client0 NetX Duo Web HTTP/HTTPs Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo Web HTTP/HTTPs Client

When the NetX Duo Web HTTP/HTTPs Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

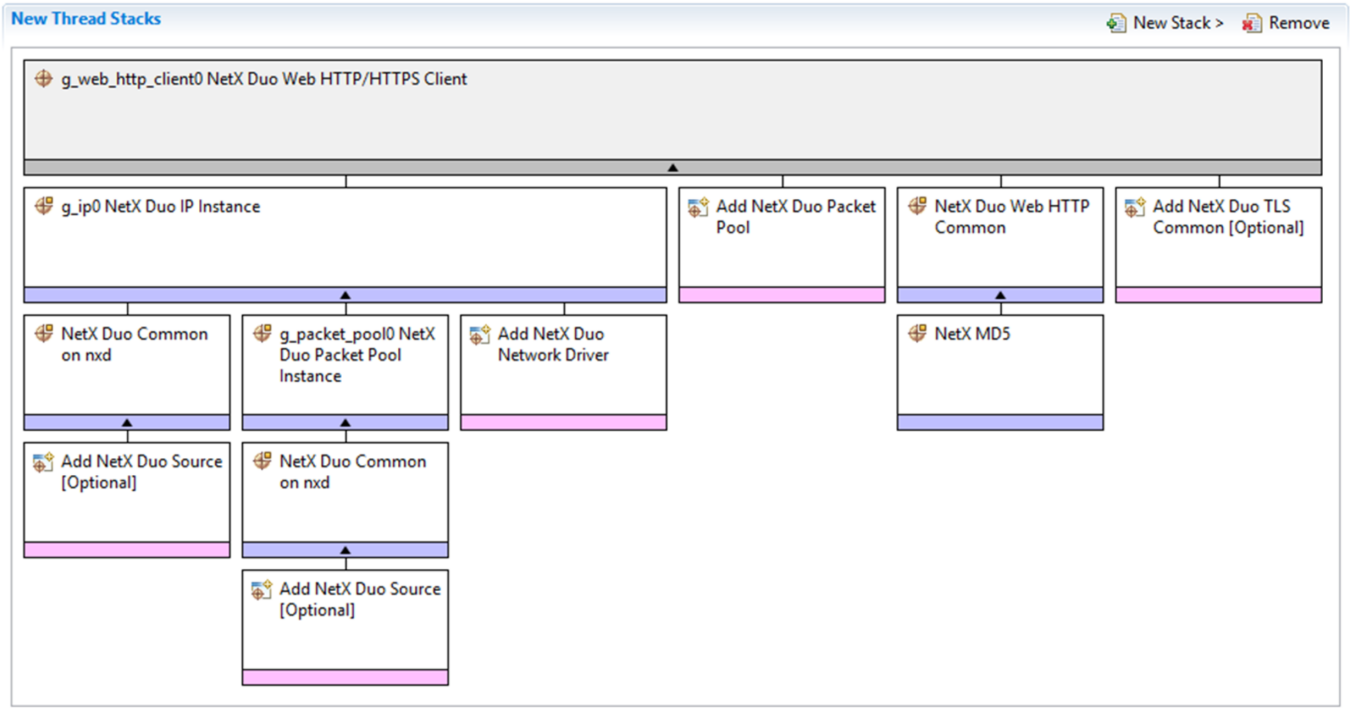


Figure 468: NetX Duo Web HTTP/HTTPs Client Module Stack

In the stack above, the NetX Duo Network Driver has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx

- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.23.5 Configuring the NetX Duo Web HTTP/HTTPs Client Module

The NetX Duo Web HTTP/HTTPs Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo Web HTTP/HTTPs Client Module

ISDE Property	Value	Description
Parameter Checking	Enable, Disable, BSP Default: BSP	Selects if code for parameter checking is to be included in the build.
Minimum packet size (bytes)	300	Select the minimum packet size in bytes.
HTTPS Support	Enable, Disable Default: Disable	Select whether to enable HTTPS support.
Name	g_web_http_client0	Module name.
TCP socket window size (bytes)	1024	Select the TCP socket window size in bytes.
Name of generated initialization	web_http_client_init0	Name of generated initialization selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different pins for the Ethernet peripheral. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo Web HTTP/HTTPs Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Disable	TCP selection
UDP	Enable, Disable Default: Disable	UDP selection
ICMP	Enable, Disable Default: Disable	ICMP selection

IGMP	Enable, Disable Default: Disable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Web HTTP Common Instance

ISDE Property	Value	Description
Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragmentation option selection.
MD5 Support	Enable, Disable Default: Disable	MD5 support selection.
Time to live	128	Time to live selection.
Maximum password length (bytes)	20	Maximum password length in bytes.
Maximum username length (bytes)	20	Maximum username length in bytes.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description

Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo Web HTTP/HTTPs Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX Duo Web HTTP/HTTPs Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.23.6 Using the NetX Duo Web HTTP/HTTPs Client Module in an Application

Once the module has been configured and the files auto generated, the NetX Web HTTP/HTTPs client Module is ready to be used in an application. Note that the auto generated code includes the initialization function with the name specified under the Name of generated initialization function property. This function internally calls the nx_web_http_client_create() API to create a HTTP/HTTPs client instance with the name specified under the Name property. Calls to this initialization function will be enabled or disabled depending the Auto Initialization property value. Once the client instance is created, the typical steps in using the module in an application are:

1. The application can connect to a plaintext HTTP server by invoking nx_web_http_client_connect(). If the application needs to use secure connection, it can connect to a secure HTTP server by invoking nx_web_http_client_secure_connect(). Note that application needs to implement the TLS setup callback function for secure connection. If needed, the application can include additional server certificate validation using DNS validation, certificate revocation and certificate policy enforcement. This can be done by invoking nx_secure_tls_session_certificate_callback_set() inside the TLS setup callback function. A reference implementation of TLS setup callback function is provided in Figure 1 of appendix.These APIs just opens a plain or secure connection to server but does not send any request.
2. After connecting to the server, the client can create a custom HTTP request such as GET by calling nx_web_http_- client_request_initialize().
3. The client can add custom header to request created in step 2 by invoking nx_web_http_client_request_header_add()API.
4. After connecting to the server, the client can send request to server using nx_web_http_client_request_send() API.
5. The application can retrieve the response sent by the server by repeatedly calling the nx_web_http_client_response_body_get() API until the entire response is retrieved.
6. The application should invoke nx_web_http_client_delete() to delete all the resources associated with HTTP client instance.

Note

If the application does not want to create custom requests, then it can use nx_web_http_*_start() APIs to send a standard request. For example, to send a standard GET request to plain text server, the application can use nx_web_http_client_get_start(). Similarly, the application can use nx_web_http_client_get_secure_start() to send a standard GET request over a secure channel to server. The nx_web_http_*_start() APIs internally uses nx_web_http_client_request_initialize() to create and send the desired HTTP request.

The following figure illustrates common steps in a typical operational flow diagram:

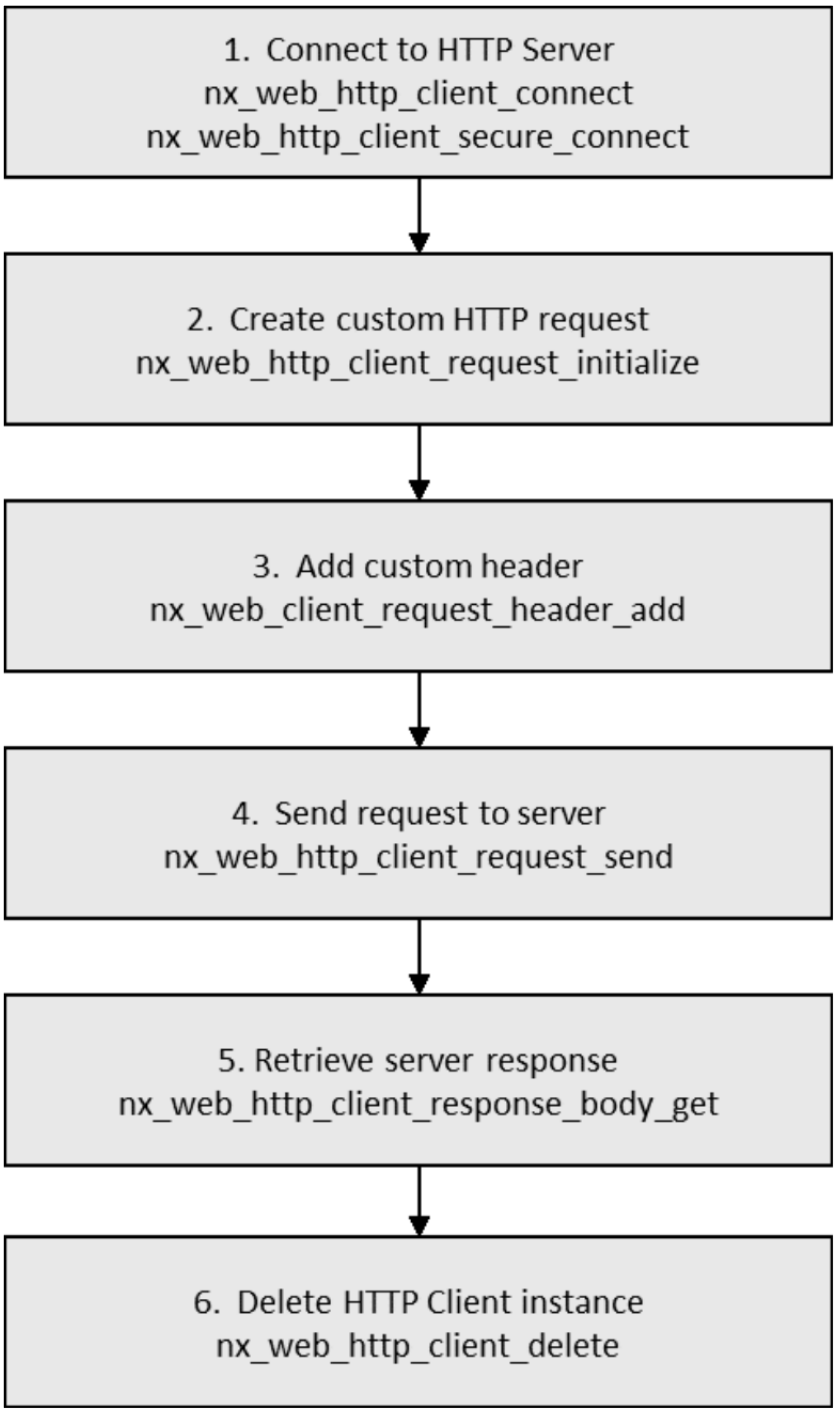


Figure 469: Flow Diagram of a Typical NetX Duo Web HTTP/HTTPs Client Module Application

4.3.24 NetX/NetX Duo HTTP/HTTPS Web Server Framework

4.3.24.1 NetX Duo Web HTTP/HTTPs Server Introduction

The NetX Duo Web HTTP/HTTPs Server module provides a high-level API for hosting Hyper Text

Transport Protocol (HTTP) Server on the web. The HTTP protocol utilizes Transmission Control Protocol (TCP) services to perform its function. HTTPs is the secure version of the HTTP protocol which uses HTTP on top of the Transport Layer Security (TLS) protocol to secure underlying TCP connection.

HTTP/HTTPs server is implemented on top of NetX Duo IP and NetX Duo Packet Pool. NetX Duo IP attaches itself to appropriate link layer driver such as Ethernet/Wi-Fi/Cellular. Web HTTP server can be started over secure connection and in such case, it uses service provided by NetX Duo TLS Common.

Unsupported Features

NetXDuo HTTPs Server has not been tested on the Cellular CAT1 and Quectel BG96 Cellular Modules.

NetX Duo Web HTTP/HTTPs Server Module Features

- NetX Web HTTP/HTTPs server is compliant with below RFCs
  - RFC1945 "Hypertext Transfer Protocol/1.0"
  - RFC 2616 "Hypertext Transfer Protocol – HTTP/1.1"
  - RFC 2818 "HTTP over TLS"
  - RFC 2581 "TCP Congestion Control"
  - RFC 1122 "Requirements for Internet Hosts", and related RFCs
- Multipart support
- Basic and digest authentication support
- Custom MIME Support
- Callback support for several key functions:
  - HTTP Authentication Callback
  - HTTP Request Notify Callback
  - HTTP Invalid Username/Password Callback
  - HTTP Insert GMT Date Header Callback
  - HTTP Cache Info Get Callback
- Provides option to enable/disable TLS for secure communication using NetX Secure in SSP

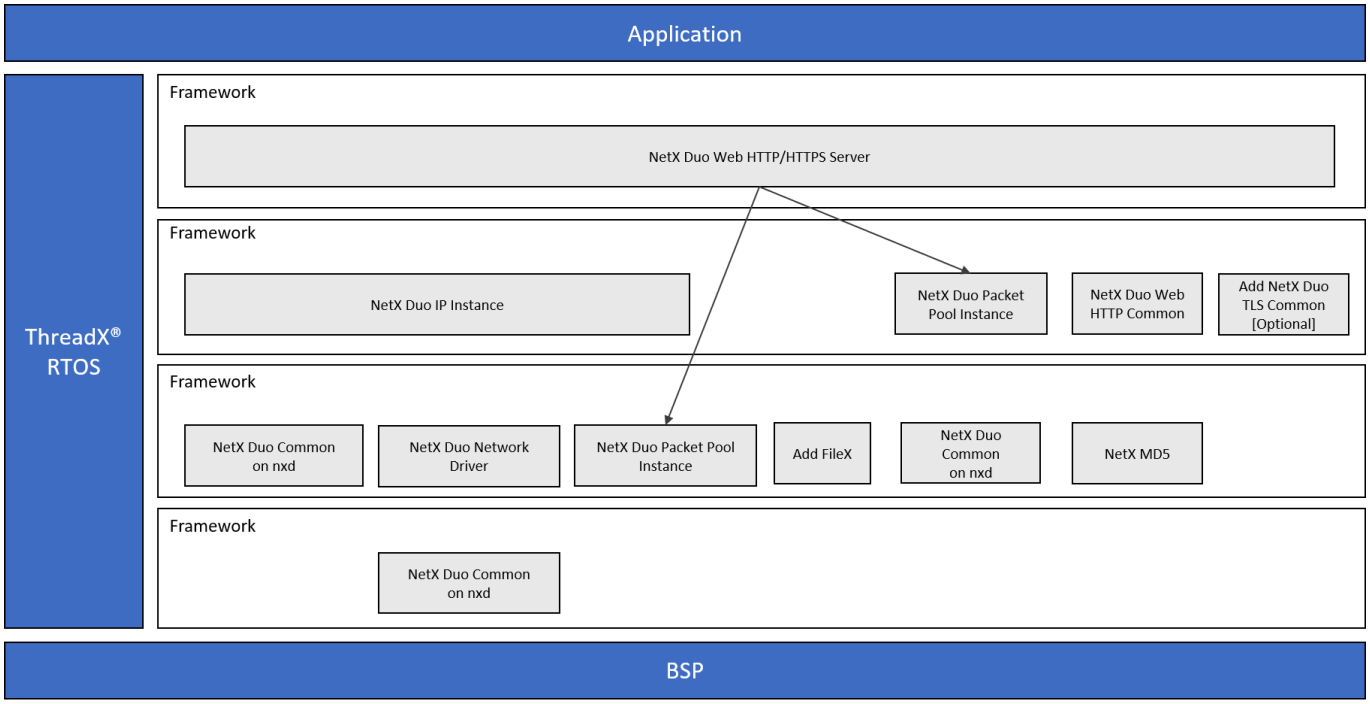


Figure 470: NetX Duo Web HTTP/HTTPs Server Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo Web HTTP/HTTPs Server Module in an Application for additional details.

4.3.24.2 NetX Duo Web HTTP/HTTPs Server Module APIs Overview

The NetX Duo Web HTTP/HTTPs Server module defines APIs for creating, deleting, starting, stopping, response sending and getting information for a received HTTP request. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo Web HTTP/HTTPs Server Module API Summary

Function Name	Example API Call and Description
nx_web_http_server_cache_info_callback_set	nx_web_http_server_cache_info_callback_set(NX_WEB_HTTP_SERVER *server_ptr, UINT (*cache_info_get)(CHAR *resource, UINT *max_age, NX_WEB_HTTP_SERVER_DATE *date)); Set the callback to retrieve URL max age and date
nx_web_http_server_callback_data_send	nx_web_http_server_callback_data_send(NX_WEB_HTTP_SERVER *server_ptr, VOID *data_ptr, ULONG data_length); Send data from callback function
nx_web_http_server_callback_generate_response_header	nx_web_http_server_callback_generate_response_header(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_pptr, CHAR *status_code, UINT content_length, CHAR *content_type, CHAR* additional_header); Create a response header in a callback function
nx_web_http_server_callback_packet_send	nx_web_http_server_callback_packet_send(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET *packet_ptr); Send an HTTP packet from callback function
nx_web_http_server_callback_response_send	nx_web_http_server_callback_response_send(NX_WEB_HTTP_SERVER *server_ptr, CHAR *header, CHAR *information, CHAR additional_info); Send response from callback function

nx_web_http_server_content_get	nx_web_http_server_content_get(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET *packet_ptr, ULONG byte_offset, CHAR *destination_ptr, UINT destination_size, UINT *actual_size); Get content from the request
nx_web_http_server_content_get_extended	nx_web_http_server_content_get_extended(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET *packet_ptr, ULONG byte_offset, CHAR *destination_ptr, UINT destination_size, UINT *actual_size); Get content from the request/supports zero length Content Length
nx_web_http_server_content_length_get	nx_web_http_server_content_length_get(NX_PACKET *packet_ptr, UINT *content_length); Get length of content in the request/supports Content Length of zero value
nx_web_http_server_create	nx_web_http_server_create(NX_WEB_HTTP_SERVER *http_server_ptr, CHAR *http_server_name, NX_IP *ip_ptr, UINT server_port, FX_MEDIA *media_ptr, VOID *stack_ptr, ULONG stack_size, NX_PACKET_POOL *pool_ptr, UINT (*authentication_check)(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type, CHAR *resource, CHAR **name, CHAR **password, CHAR **realm), UINT (*request_notify)(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type, CHAR *resource, NX_PACKET *packet_ptr)); Create an HTTP Server instance
nx_web_http_server_delete	nx_web_http_server_delete(NX_WEB_HTTP_SERVER *http_server_ptr); Delete an HTTP Server instance
nx_web_http_server_get_entity_content	nx_web_http_server_get_entity_content(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_pptr, ULONG *available_offset, ULONG *available_length); Retrieve the location and length of entity data
nx_web_http_server_get_entity_header	nx_web_http_server_get_entity_header(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_pptr, UCHAR *entity_header_buffer, ULONG buffer_size); Retrieve the contents of entity header

nx_web_http_server_gmt_callback_set	nx_web_http_server_gmt_callback_set(NX_WEB_HTTP_SERVER *server_ptr, VOID (*gmt_get)(NX_WEB_HTTP_SERVER_DATE *date); Set the callback to obtain GMT date and time
nx_web_http_server_invalid_userpassword_notify_set	nx_web_http_server_invalid_userpassword_notify_set(NX_WEB_HTTP_SERVER *http_server_ptr, UINT (*invalid_username_password_callback)(CHAR *resource, ULONG client_address, UINT request_type)); Set the callback to handle invalid user/password
nx_web_http_server_mime_maps_additional_set	nx_web_http_server_mime_maps_additional_set(NX_WEB_HTTP_SERVER *server_ptr, NX_WEB_HTTP_SERVER_MIME_MAP *mime_maps, UINT mime_maps_num); Set additional MIME maps for HTML
nx_web_http_server_response_packet_allocate	nx_web_http_server_response_packet_allocate(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_ptr, ULONG wait_option); Allocate a HTTP(S) packet
nx_web_http_server_packet_content_find	nx_web_http_server_packet_content_find(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_ptr, UINT *content_length); Extract content length and set pointer to start of data
nx_web_http_server_packet_get	nx_web_http_server_packet_get(NX_WEB_HTTP_SERVER *server_ptr, NX_PACKET **packet_ptr); Receive the next HTTP packet
nx_web_http_server_param_get	nx_web_http_server_param_get(NX_PACKET *packet_ptr, UINT param_number, CHAR *param_ptr, UINT *param_size, UINT max_param_size); Get parameter from the request
nx_web_http_server_query_get	nx_web_http_server_query_get(NX_PACKET *packet_ptr, UINT query_number, CHAR *query_ptr, CHAR *query_size, UINT max_query_size); Get query from the request
nx_web_http_server_response_chunked_set	nx_web_http_server_response_chunked_set(NX_WEB_HTTP_SERVER *server_ptr, UINT chunk_size, NX_PACKET *packet_ptr); Set chunked transfer for HTTP(S) response

nx_web_http_server_secure_configure	nx_web_http_server_secure_configure(NX_WEB_HTTP_SERVER *http_server_ptr, const NX_SECURE_TLS_CRYPTO *crypto_table, VOID *metadata_buffer, ULONG metadata_size, UCHAR* packet_buffer, UINT packet_buffer_size, NX_SECURE_X509_CERT *identity_certificate, NX_SECURE_X509_CERT *trusted_certificates[], UINT trusted_certs_num, NX_SECURE_X509_CERT *remote_certificates[], UINT remote_certs_num, UCHAR *remote_certificate_buffer, UINT remote_cert_buffer_size); Configure an HTTP Server to use TLS for secure HTTPS *Note* *Parameter crypto_table is a pointer to crypto cipher suite table. Suggested ciphersuite to be used is "nx_crypto_tls_ciphers_synergys7" which is a cipher suite table under NetX in SSP. For further information on Ciphersuite Refer NetX TLS User Manual.*
nx_web_http_server_start	nx_web_http_server_start(NX_WEB_HTTP_SERVER *http_server_ptr); Start the HTTP Server
nx_web_http_server_stop	nx_web_http_server_stop(NX_WEB_HTTP_SERVER *http_server_ptr); Stop the HTTP Server
nx_web_http_server_type_get	nx_web_http_server_type_get(NX_WEB_HTTP_SERVER *http_server_ptr, CHAR *name, CHAR *http_type_string, UINT *string_size); Extract file type from Client HTTP request

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful connection of TCP socket.
NX_PTR_ERROR	Invalid pointer input.
NX_WEB_HTTP_DATA_END	End of request content
NX_WEB_HTTP_POOL_ERROR	Invalid payload size in packet pool
NX_CALLER_ERROR	Invalid caller of this service.

NX_WEB_HTTP_TIMEOUT	HTTP Server timeout in getting next packet of content
NX_WEB_HTTP_ERROR	Internal HTTP Server error.
NX_WEB_HTTP_FAILED	Query size too small
NX_CALLER_ERROR	Invalid caller of this service
NX_WEB_HTTP_BOUNDARY_ALREADY_FOUND	Content for the HTTP server internal multipart markers is already found
NX_WEB_HTTP_NOT_FOUND	Entity header field/query not found
NX_NO_PACKET	No packet available
NX_WAIT_ABORTED	Requested suspension was aborted
NX_WEB_HTTP_INCOMPLETE_PUT_ERROR	Improper HTTP header format
NX_INVALID_PARAMETERS	Packet size cannot support protocol
NX_WEB_HTTP_IMPROPERLY_TERMINATED_PARAM	Request parameter not properly terminated
NX_WEB_HTTP_NO_QUERY_PARSED	No query in Client request
NX_NOT_CONNECTED	The underlying TCP socket is no longer connected.
NX_SECURE_TLS_UNRECOGNIZED_MESSAGE_TYPE	A received TLS message type is incorrect.
NX_SECURE_TLS_UNSUPPORTED_CIPHER	Cipher provided by the remote host is not supported.
NX_SECURE_TLS_HANDSHAKE_FAILURE	Message processing during the TLS handshake has failed.
NX_SECURE_TLS_HASH_MAC_VERIFY_FAILURE	An incoming message failed a hash MAC check.
NX_SECURE_TLS_TCP_SEND_FAILED	An underlying TCP socket send failed.
NX_SECURE_TLS_INCORRECT_MESSAGE_LENGTH	An incoming message had an invalid length field.
NX_SECURE_TLS_BAD_CIPHERSPEC	An incoming ChangeCipherSpec message was incorrect.
NX_SECURE_TLS_INVALID_SERVER_CERT	An incoming TLS certificate is unusable for identifying the remote TLS server.
NX_SECURE_TLS_UNSUPPORTED_PUBLIC_CIPHER	The public-key cipher provided by the remote host is unsupported
NX_SECURE_TLS_NO_SUPPORTED_CIPHERS	The remote host has indicated no ciphersuites that are supported by the NetX Secure TLS stack.
NX_SECURE_TLS_UNKNOWN_TLS_VERSION	A received TLS message had an unknown TLS version in its header.

NX_SECURE_TLS_UNSUPPORTED_TLS_VERSION	A received TLS message had a known but unsupported TLS version in its header.
NX_SECURE_TLS_ALLOCATE_PACKET_FAILED	An internal TLS packet allocation failed.
NX_SECURE_TLS_INVALID_CERTIFICATE	The remote host provided an invalid certificate.
NX_SECURE_TLS_ALERT_RECEIVED	The remote host sent an alert indicating an error and ending the TLS session.
NX_SECURE_TLS_MISSING_CRYPTO_ROUTINE	An entry in the ciphersuite table had a NULL function pointer.
NX_WEB_HTTP_EXTENSION_MIME_DEFAULT	Default "text/plain" returned

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.24.3 NetX Duo Web HTTP/HTTPs Server Module Operational Overview

The NetX Duo Web HTTP Server module creates an IP instance that carries out NetX Duo operations and enables it for TCP services in the NetX Duo library; it then creates the Web HTTP Server instance and TCP socket for listening to client requests on configured port number. The Web HTTP Server requires a packet pool; the module can supply one either by sharing the IP default packet pool (g_packet_pool0) or create a new one. The minimum packet payload is set by the Minimum size of packets in pool property of the Web HTTP Server module. This packet pool is used by the Web HTTP Server only to transmit packets, so the packet pool size and payload can be optimized on the expected size and number of HTTP Server packets sent out.

The NetX Duo Web HTTP Server supports both IPv4 and IPv6 connections. If the HTTP Server has clients desiring to connect over IPv6, make sure the NetX Duo IPv6 Support property is enabled in the NetX Duo Source element. It may be necessary to enable ICMPv6 checksum computation for the underlying ICMPv6 protocols. To do so, set the Checksum computation support on transmitted ICMPv6 packets and Checksum computation support on received ICMPv6 packets properties of the NetX Duo Source element to Enabled. (If the host hardware automatically computes ICMPv6 checksums, these can be left disabled.) Make sure the IPv6 Global Address of the Client host is set in the IP instance element. Thereafter, the NetX Duo does the necessary processing to enable IPv6 and ICMPv6 services required for IPv6 underlying protocols.

The NetX Duo Web HTTP Server is also designed for use with the FileX embedded file system.

The NetX Duo Web HTTP/HTTPs server module can be used in the normal mode or secure mode

NetX Duo Web HTTP/HTTPs server module Normal Mode Operational Description

In normal mode, the communication between HTTP client and server is not secure.

Netx Duo Web HTTP/HTTPs server module Secure Mode Operational Description

In Secure mode, the communication between HTTP client and server is secured using the TLS protocol. In the thread pane, TLS protocol is represented by "Add NetX Duo TLS common [Optional]" block as shown in the figure below

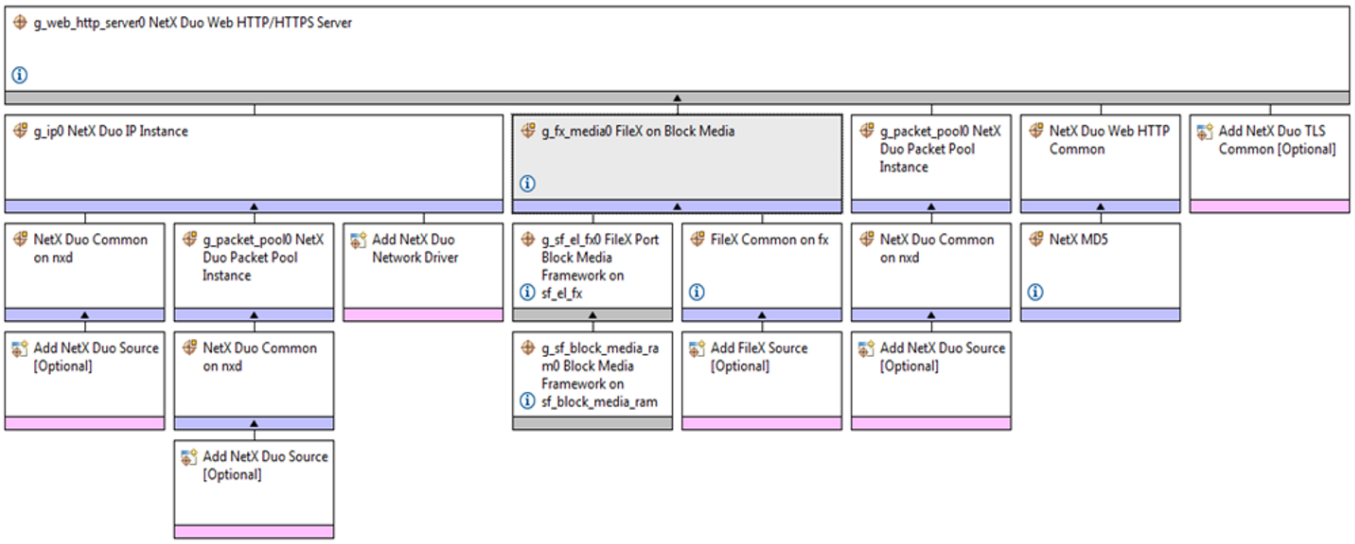


Figure 471: NetX Duo Web HTTP/HTTPs Server Module Component Thread-Pane View

Adding NetX Duo TLS Common block enables TLS support. The figure below shows the thread pane view of Web HTTP server with TLS support enabled.

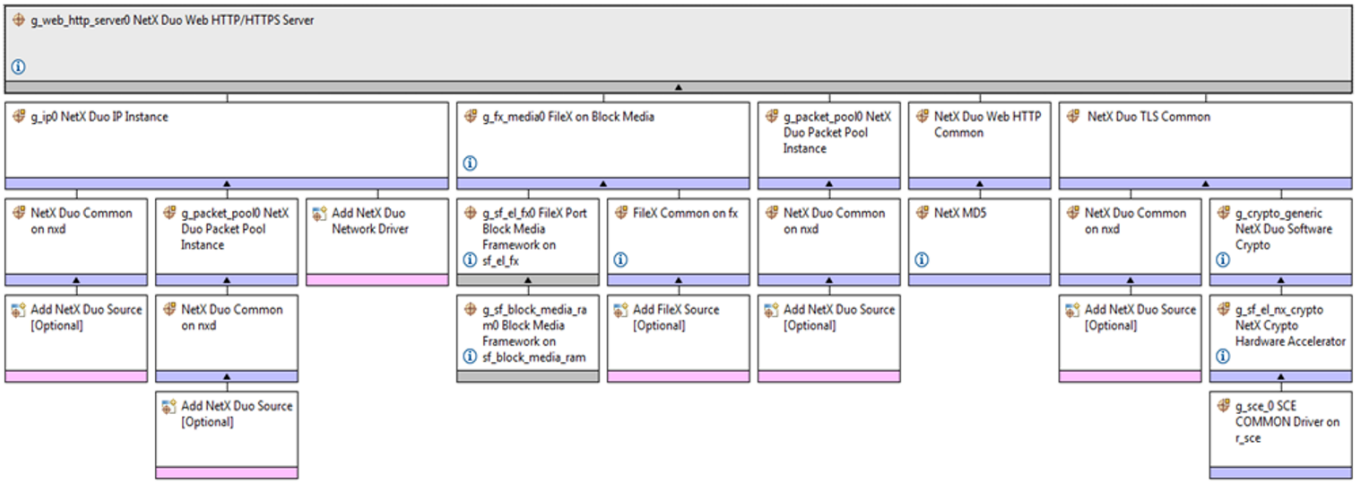


Figure 472: NetX Duo Web HTTP/HTTPs Server Module Component with TLS Support Enabled

NetX Duo Web HTTP Server Responses

When the Web HTTP Server processes the client command, it returns an ASCII response string that includes a 3-digit numeric status code. The numeric response is used by the HTTP Client software to determine whether the operation succeeded or failed. Following is a list of various HTTP Server responses to client commands:

Web HTTP Server responses to client commands

Numeric Field	Meaning
200	Request was successful

400	Request was not formed properly
401	Unauthorized request, client needs to send authentication
404	Specified resource in request was not found
500	Internal HTTP Server error
501	Request not implemented by HTTP Server
502	Service is not available

For example, a successful client request to PUT the file test.htm is responded to with the message HTTP/1.0 200 OK.

NetX Duo Web HTTP Authentication

HTTP authentication is optional and is not required for all web requests. There are two types of authentication, basic and digest. Basic authentication is equivalent to the name and password authentication found in many protocols. In HTTP basic authentication, the name and passwords are concatenated and encoded in the base64 format. The main disadvantage of basic authentication is the name and password are transmitted openly in the request, making it easy for the name and password to be stolen. Digest authentication addresses this problem by never transmitting the name and password in the request. Instead, an algorithm is used to derive a 128-bit key or digest from the name, password, and other information. The NetX Web HTTP Server supports the standard MD5 digest algorithm.

The Web HTTP Server authentication callback can decide if a requested resource requires authentication. If authentication is required and the client request did not include the proper authentication, an HTTP/1.0 401 Unauthorized response with the type of authentication required is sent to the client. The client is then expected to form a new request with the proper authentication.

NetX Duo Web HTTP Authentication Callback

The Web HTTP Server authentication callback routine is specified by the Name of Authentication Checking Function property of the HTTP Server Thread. This function is called at the beginning of each HTTP Client request.

The callback routine provides the NetX Web HTTP Server with the username, password, and realm strings associated with the resource and returns the type of authentication necessary. If no authentication is necessary for the resource, the authentication callback should return the value of NX_WEB_HTTP_DONT_AUTHENTICATE. If basic authentication is required for the specified resource, the routine should return NX_WEB_HTTP_BASIC_AUTHENTICATE. If MD5 digest authentication is required, the callback routine should return NX_WEB_HTTP_DIGEST_AUTHENTICATE.

The format of the authenticate callback routine is defined as:

UINT nx_web_http_server_authentication_check(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type, CHAR *resource, CHAR **name, CHAR **password, CHAR **realm);

The input parameters are defined as follows:

Input Parameters Definitions

Parameter	Meaning

request_type	Specifies the HTTP Client request, valid requests are defined as: NX_WEB_HTTP_SERVER_GET_REQUEST NX_WEB_HTTP_SERVER_POST_REQUEST NX_WEB_HTTP_SERVER_HEAD_REQUEST NX_WEB_HTTP_SERVER_PUT_REQUEST NX_WEB_HTTP_SERVER_DELETE_REQUEST
resource	Specific resource requested.
name	Destination for the pointer to the required username.
password	Destination for the pointer to the required password.
realm	Destination for the pointer to the realm for this authentication.

Name, password, and realm pointers are not used if NX_WEB_HTTP_DONT_AUTHENTICATE is returned by the authentication callback routine. The HTTP Server developer must ensure that the maximum size of the username and password (defined by the Maximum username length and Maximum password length properties of the NetX Duo Web HTTP Common) are large enough for the username and password specified in the authentication callback. These are both defaulted to size 20 characters.

NetX Duo Web HTTP Server Request Notify callback

If a request callback is specified, (the Name of Request Notify Callback Function property of the NetX Web HTTP Server module) the NetX Web HTTP Server forwards requests to the specified function after authentication and validity of the client request is completed without errors. The callback should indicate (by the return status) if no more processing of the client request is required (return status NX_WEB_HTTP_CALLBACK_COMPLETED), if there was an error in the callback processing, (status is non-zero), or the process was completed successfully, and the Web HTTP Server should continue processing the client request. The format of this callback is:

UINT request_notify(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type, CHAR *resource, NX_PACKET *packet_ptr);

To disable the request notify callback, set the Name of Request Notify Callback Function property to NULL.

NetX Duo Web HTTP Invalid Username/Password Callback

The optional Invalid Username/Password callback in the NetX Web HTTP Server module is invoked if the HTTP Server receives an invalid username-and-password combination in a client request. To set the Invalid Username/Password callback function, use the nx_web_http_server_invalid_userpassword_notify_set service.

NetX Duo Web HTTP Insert GMT Date Header Callback

The NetX Web HTTP Server supports an optional callback to insert a date header in its response messages. This callback is invoked when the Server responds to a Client PUT or GET request. To register a GMT date callback with the HTTP Server, the nx_web_http_server_gmt_callback_set service is used.

NetX Duo Web HTTP Cache Info Get Callback

The NetX Web HTTP Server has an optional callback to request the maximum age and date from the HTTP application for a specific resource. This information is used to determine if the HTTP server sends the entire page in response to a Client Get request. If the if modified since in the Client request is not found or does not match the last modified date returned by the get-cache callback, the entire page is sent. To register the callback with the HTTP server nx_web_http_server_cache_info_callback_set service is used.

NetX Duo Web HTTP Multipart Support

Multipurpose Internet Mail Extensions (MIME) was originally intended for the SMTP protocol, but its use has spread to HTTP. MIME allows messages to contain mixed message types (for example, image/jpg and text/plain) within the same message. The NetX Web HTTP Server has added services to determine content type in HTTP messages containing MIME from the client. To enable multipart support, set the Multipart HTTP requests support property of the NetX Web HTTP Server module to enable.

NetX Duo Web HTTP/HTTPs Server Module Important Operational Notes and Limitations

NetX Duo Web HTTP/HTTPs Server Module Operational Notes

- The NetX Duo Web HTTP Server module requires a FileX media (Block media or USB Mass Storage). When an HTTP Server stack element is added to the project, an Add FileX box is attached to it. The configurator automatically sets up and initializes the FileX media for the server before the server is started. For more details for configuring FileX, see *FileX User's Guide for the Renesas Synergy Platform*.
- The NetX Duo Web HTTP Server also requires a packet pool for transmitting packets. It can share the IP default packet pool or create a separate packet pool.
- The NetX Duo Web HTTP/HTTPs Server component is added by clicking on the (+) sign in the thread pane window -> Azure RTOS -> NetX Duo -> Protocols -> NetX Duo Web HTTP/HTTPs Server
- Adding the NetX Duo Web HTTP/HTTPs Server component to a project automatically adds the option to add the NetX Duo TLS component required for secure HTTP server

The NetX Duo Web HTTP/HTTPs Server properties are listed in the following table:

Property	Value
◢ Common	
Parameter Checking	Default (BSP)
HTTPS Support	Disable
FileX Support	Enable
Multipart HTTP requests support	Disable
Server thread priority	16
Server thread time slicing interval (ticks)	2
Server socket window size (bytes)	2048
Server time out (seconds)	10
Server time out for accept (seconds)	10
Server time out for disconnect (seconds)	10
Server time out for receive (seconds)	10
Server time out for send (seconds)	10
Maximum size of header field (bytes)	256
Maximum connections in queue	4
Maximum length of resource name	40
Number of simultaneous sessions for server	2
Minimum size of packets in pool (bytes)	600
Maximum number of queued transmit packets (units)	20
Server Socket Re-transmission Timeout (seconds)	2
Maximum number of retries per packet	10
Server Next Re-transmission timeout shift	1
◢ Module g_web_http_server0 NetX Duo Web HTTP/HTTPS Server	
Name	g_web_http_server0
Internal thread stack size (bytes)	4096
TCP listening port for HTTP/HTTPS Server	80
Name of Authentication Checking Callback Function	NULL
Name of Request Notify Callback Function	NULL
Name of generated initialization function	web_http_server_init0
Auto Initialization	Enable

Figure 473: NetX Duo Web HTTP/HTTPs Server Module Component Configurable Properties

In the figure above, "Common" properties are those configurable options in the NetX Duo Web HTTP/HTTPs Server that are common to all instances of the HTTP/HTTPs server in the project. The "Module" properties are specific to each instance of HTTP/HTTPs Server in the project.

NetX Duo Web HTTP/HTTPs Server Module Common Properties

- Parameter Checking:This option removes the basic HTTP error checking. It is typically used after the application has been debugged. Default value is BSP.
- HTTPS Support: If defined, this macro enables TLS and HTTPS. Leave undefined to free up resources if only plaintext HTTP is desired. Default value is Disable.
- FileX Support: This option provides User to select either "FileX Stub", "FileX on USB Mass Storage" or "FileX on Block Media". Default value is Enable.
- Multipart HTTP requests support: If defined, this option enables HTTP Server to support multipart HTTP requests. Default value is Disable.
- Server Thread Priority: The priority of the HTTP Server thread. Default value is 16.
- Server Thread time slicing interval (ticks): The number of timer ticks the Server thread is allowed to run before yielding to threads of the same priority. Default value is 2 ticks.
- Server socket window size (bytes): Size of the Server TCP socket receive window. Default value is 2048 bytes.
- Server time out (seconds): Time period for which internal services of server will suspend. Default value is set to 10 seconds (10 * NX_IP_PERIODIC_RATE).
- Server time out for accept (seconds): Time period for which internal services will suspend for in internal nx_tcp_server_socket_accept() calls. The default value is set to 10 seconds (10 * NX_IP_PERIODIC_RATE).
- Server time out for disconnect (seconds): Specifies the number of ThreadX ticks that internal services will suspend for in internal nx_tcp_socket_disconnect() calls. The default

value is set to 10 seconds (10 * NX_IP_PERIODIC_RATE)
- Server time out for receive (seconds): Specifies the number of ThreadX ticks that internal services will suspend for in internal nx_tcp_socket_receive() calls. The default value is set to 10 seconds (10 * NX_IP_PERIODIC_RATE)
- Server time out for send (seconds): Specifies the number of ThreadX ticks that internal services will suspend for in internal nx_tcp_socket_send() calls. The default value is set to 10 seconds (10 * NX_IP_PERIODIC_RATE)
- Maximum size of header field (bytes): Specifies the maximum size of the HTTP header field. Default value is 256 bytes.
- Maximum connections in queue: Specifies the number of connections that can be queued for the HTTP Server. The default value is 4 (twice the maximum number of server sessions).
- Maximum length of resource name: Specifies the length of the resource name. Default value is 40.
- Number of simultaneous sessions for server: Specifies the number of simultaneous sessions for an HTTP or HTTPS server. A TCP socket and a TLS session (if HTTPS is enabled) are allocated for each session. Default value is 2.
- Minimum size of packets in pool (bytes): Specifies the minimum size of the packets in the pool specified at Server creation. The minimum size is needed to ensure the complete HTTP header can be contained in one packet. Default value is 600 bytes.
- Maximum number of queued transmit packets (units): This option specifies the maximum number of packets that can be enqueued on the Server socket re-transmission queue. If the number of packets enqueued reaches this number, no more packets can be sent until one or more enqueued packets are released. Default value is 20 units.
- Server Socket Re-transmission Timeout (seconds): Timeout for Server socket re-transmission. Default value is 2 seconds.
- Maximum number of retries per packet: This parameter sets the maximum number of re-transmissions on Server socket. Default value is 10.
- Server Next Re-transmission timeout shift: This value is used to set the next re-transmission timeout. The current timeout is multiplied by the number of re-transmissions thus far, shifted by the value of the socket timeout shift. Default value is 1 for doubling the timeout

NetX Duo Web HTTP/HTTPs Server Module Properties

- Name: Name of Web HTTP/HTTPs server instance. Default value is g_web_http_server0
- Internal Thread stack size (bytes): HTTP Server thread stack size. Used as a parameter while creating web HTTP Server instance. Default value is 4096 bytes
- TCP listening port for HTTP/HTTPS Server: TCP listening port for secured/unsecured HTTP server instance. Default value is 80.

Note: *If user wants to start Web HTTP server in secured mode then user can set the port number accordingly using this property. For example, user can set 443 as port number for secured HTTP server and can set port number as 80 for unsecured HTTP server.*

- Name of authentication checking callback function: Application's authentication checking routine. If specified, this routine is called for each HTTP Client request. If this parameter is NULL, no authentication will be performed. Default value is NULL.
- Name of request notify callback function: Application's request notify routine. If specified, this routine is called prior to the HTTP server processing of the request. This allows the resource name to be redirected or fields within a resource to be updated prior to completing the HTTP Client request. Default value is NULL.
- Name of generated initialization function: Name of initialization function under whichweb HTTP Server instance is created. Default value is web_http_server_init0.
- Auto Initialization: Enable/disable call to initialization function. This determines if the function specified in the Name of Generated Initialization function option is called. If set to Enable, it will invoke this function. Otherwise if set to Disable, the application must call the

nx_web_http_server_create() API before using any NetX Web HTTP/HTTPs Server services.

NetX Duo Web HTTP Common Configurable Properties

The common properties are listed in the following table:

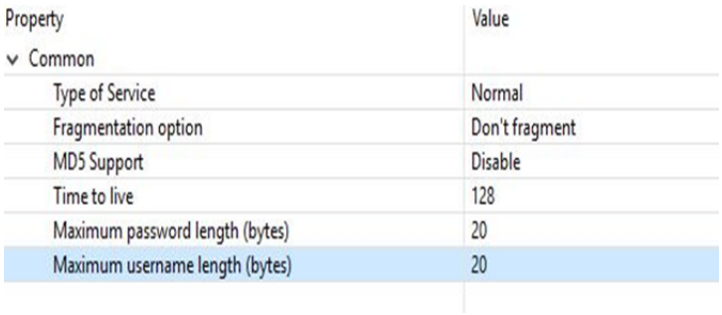


Figure 474: NetX Duo Web HTTP/HTTPs Client Module Component Common Configurable Properties

- Type of Service: Type of network service required for the HTTP TCP requests. The network service can be of type (Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost) and the default value is Normal.
- Fragmentation Option: This enables/disables TCP fragmentation for HTTP client request. The default value is Don't Fragment.
- MD5 Support: This enables/disables MD5 digest support required for digest authentication. The default value is disable.
- Time to live: This specifies maximum no. of routers the HTTP packet can pass through before it get discarded. The default is 128.
- Maximum password length (bytes): This defines the maximum length of client supplied password. The default is 20.
- The Maximum username length (bytes): This defines the maximum length of client supplied username. The default is 20.

NetX Duo Web HTTP/HTTPs Server Module Limitations

- TRACE, OPTIONS, and CONNECT HTTP requests are not supported.
- The HTTP Server supports both basic and MD5 digest authentication, but not MD5-sess.
- No content compression is supported.
- The packet pool associated with the HTTP Server must be large enough to hold the complete HTTP header.
- Request pipelining is not supported

4.3.24.4 Including the NetX Duo Web HTTP/HTTPs Server Module in an Application

This section describes how to include either or both the NetX Duo Web HTTP/HTTPs Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo Web HTTP/HTTPs Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo Web HTTP/HTTPs Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_web_http_server0 NetX Duo WebHTTP/HTTPsServer	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo Web HTTP/HTTPsServer

When the NetX Duo Web HTTP/HTTPs Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

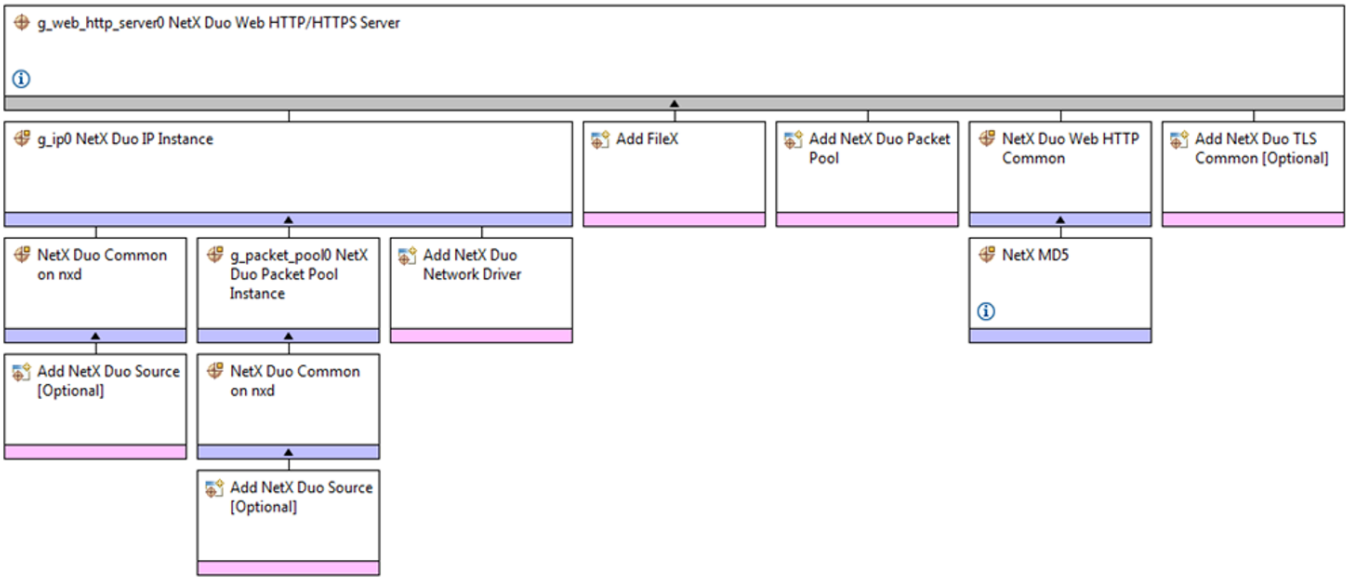


Figure 475: NetX Duo Web HTTP/HTTPs Server Module Stack

In the stack above, the NetX Duo Network Driver has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

Additionally, in the stack above, the FileX stack has also not been populated yet. There are multiple possible selections for the FileX module; they are not all provided so as not to needlessly complicate

the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- FileX Stub
- FileX on Block Media (implemented on Block Media Framework on sf_block_media_ram)
- FileX on USB Mass Storage (implemented on USBX Host Class Mass Storage)

4.3.24.5 Configuring the NetX Duo Web HTTP/HTTPs Server Module

The NetX Duo Web HTTP/HTTPs Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo Web HTTP/HTTPs Server Module

ISDE Property	Value	Description
Parameter Checking	Enable, Disable, BSP Default: BSP	Selects if code for parameter checking is to be included in the build.
HTTPS Support	Enable, Disable Default: Disable	Select whether to enable HTTPS support.
FileX Support	Enable, Disable Default: Enable	This option provides User to select either "FileX Stub", "FileX on USB Mass Storage" or "FileX on Block Media"
Multipart HTTP requests support	Enable, Disable Default: Disable	Option enables HTTP Server to support multipart HTTP requests
Server Thread Priority	16	Priority of the HTTP Server thread
Server Thread time slicing interval (ticks)	2	Number of timer ticks the Server thread is allowed to run before yielding to threads of the same priority.
Server socket window size (bytes)	2048	Size of the Server TCP socket receive window
Server time out (seconds)	10	Server time out selection

Server time out for accept (seconds)	10	Server time out for accept selection
Server time out for disconnect (seconds)	10	Server time out for disconnect selection
Server time out for receive (seconds)	10	Server time out for receive selection
Server time out for send (seconds)	10	Server time out for send selection
Maximum size of header field (bytes)	256	maximum size of the HTTP header field
Maximum connections in queue	4	Number of connections that can be queued for the HTTP Server
Maximum length of resource name	40	Length of the resource name
Number of simultaneous sessions for server	2	Number of simultaneous sessions for an HTTP or HTTPS server. A TCP socket and a TLS session (if HTTPS is enabled) are allocated for each session.
Minimum size of packets in pool (bytes)	600	Minimum size of the packets in the pool specified at Server creation.
Maximum number of queued transmit packets (units)	20	Specifies the maximum number of packets that can be enqueued on the Server socket re-transmission queue.
Server Socket Re-transmission Timeout (seconds)	2	Timeout for Server socket re-transmission
Maximum number of retries per packet	10	Maximum number of re-transmissions on Server socket.
Server Next Re-transmission timeout shift	1	This value is used to set the next re-transmission timeout. The current timeout is multiplied by the number of re-transmissions thus far, shifted by the value of the socket timeout shift.
Name	g_web_http_server0	Name of Web HTTP server instance
Internal Thread stack size (bytes)	4096	HTTP Server thread stack size
TCP listening port for HTTP/HTTPS Server	80	TCP listening port for secured/unsecured HTTP server instance

Name of authentication checking callback function	NULL	Name of Authentication Checking Function selection
Name of request notify callback function	NULL	Name of Request notify Function selection
Name of generated initialization function	web_http_server_init0	Name of initialization function under which web HTTP Server instance is created
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S5D9 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different pins for the Ethernet peripheral. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo Web HTTP/HTTPs Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection

IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP cache storage units	Bytes, Entries Default: Bytes	ARP cache storage units selection
ARP cache size (in storage units)	520	ARP Cache Size in Bytes/Entries selection Note: 1 Entry = 52 Bytes
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S5D9 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Web HTTP Common Instance

ISDE Property	Value	Description
Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service selection.
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragmentation option selection.
MD5 Support	Enable, Disable Default: Disable	MD5 support selection.
Time to live	128	Time to live selection.
Maximum password length (bytes)	20	Maximum password length in bytes.
Maximum username length (bytes)	20	Maximum username length in bytes.

Note

The example settings and defaults are for a project using the S5D9Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S5D9Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1568	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S5D9Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo FileX on Block Media

ISDE Property	Value	Description
Name	g_fx_media0	Module name
Format media during initialization	Enable, Disable Default: Enabled	Enabling this will format the media
File System is on block media	True, False Default: True	Whether or not the media as filesystem support
Volume Name	Volume 1	Volume name string
Number of FATs	1	Number of FATs in the media
Directory Entries	256	Number of directory entries in the root directory
Hidden Sectors	0	Number of sectors hidden before this media's boot sector
Total Sectors	3751936	Total number of sectors in the media
Bytes per Sector	512	Number of bytes per sector
Sectors per Cluster	1	Number of sectors in each cluster
Working media memory size	512	Memory allocated for file system
Name of generated initialization function	fx_media_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S5D9Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5 Instance

ISDE Property	Value	Description

No configurable properties		

Note

The example settings and defaults are for a project using the S5D9 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

4.3.24.6 Using the NetX Duo Web HTTP/HTTPs Server Module in an Application

Once the module has been configured and the files auto generated, the NetX Web HTTP/HTTPs server module is ready to be used in an application. Note that the auto generated code includes the initialization function with the name specified under the Name of generated initialization function property. This function internally calls the nx_web_http_server_create() API to create a HTTP/HTTPs server instance with the name specified under the Name property. Calls to this initialization function will be enabled or disabled depending on the Auto Initialization property value. Once the server instance is created, the typical steps in using the module in an application are:

1. Wait for valid IP address using the nx_ip_status_check API.
2. Start the Web HTTP Server.
    a. If user wants to start a secured HTTP server i.e HTTPS, the application should initialize device certificates by invoking the nx_secure_x509_certificate_initialize() API. The application should then configure an HTTP server to use TLS for secure HTTPS by invoking the nx_web_http_server_secure_configure() API. The application can start HTTPS server by invoking the nx_web_http_server_start() API.
    b. If user wants plain HTTP Server, application can start HTTP server by just invoking nx_web_http_server_start() API.
3. Handle optional callbacks if registered with the HTTP Server (Authentication Check, Request Notify, GMT set, Cache get and Invalid Username).
    a. The *authentication_check* callback routine provides the NetX Duo Web HTTP Server with the username, password, and realm strings associated with the resource and return the type of authentication necessary. If this callback is set to NULL, no authentication will be performed.
    b. The *request_notify* routine is called prior to the HTTP server processing of the request. This allows the resource name to be redirected or fields within a resource to be updated prior to completing the HTTP Client request.
4. Once the HTTP/HTTPS server is started, HTTP client can send GET/PUT/POST/HEAD/DELETE requests to the Web HTTP/HTTPS server.
5. The application should invoke nx_web_http_server_stop() to stop the server.
6. The application should invoke nx_web_http_server_delete () to delete all the resources associated with Web HTTP server instance.

Users do not have to worry about auto-generated code. Auto-generated code is included once the user generates the project after configuring the stack. Users only need to write the user application code in the associated file.

The following figure illustrates common steps in a typical operational flow diagram:

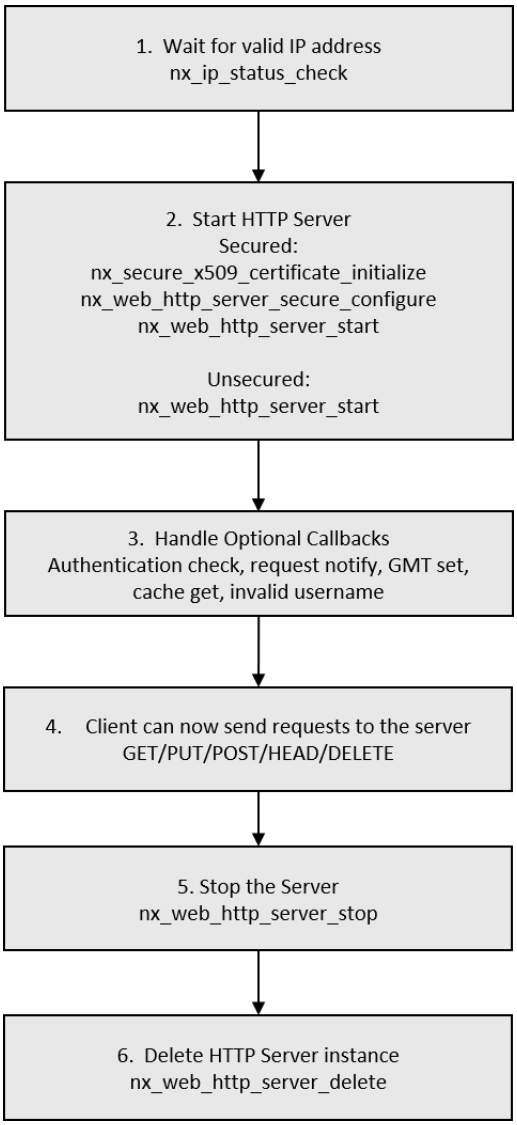


Figure 476: Flow Diagram of a Typical NetX Duo Web HTTP/HTTPs Server Module Application

4.3.25 NetX/NetX Duo SMTP Client

4.3.25.1 NetX/NetX Duo SMTP Client Introduction

The Simple Mail Transfer Protocol (SMTP) is a protocol for sending mail across networks and the Internet and utilizes the reliable Transmission Control Protocol (TCP) services to perform its content transfer function.

Note

Except for internal processing, the NetX Duo™ SMTP Client module is identical in the application, set-up and running of an SMTP Client session as the NetX™ SMTP Client module. For setting up the IP instance for IPv6 in NetX Duo, please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform.

Unsupported Features

Multiple network interface has not been tested for NetX Duo in this version of SSP.

NetX/NetX Duo SMTP Client Module Features

- The NetX SMTP Client API is compliant with RFC2821 "Simple Mail Transfer Protocol" and RFC 2554 "SMTP Service Extension for Authentication."
- Provides high-level APIs for:
  - Creating and deleting an SMTP Client
  - Sending mail messages

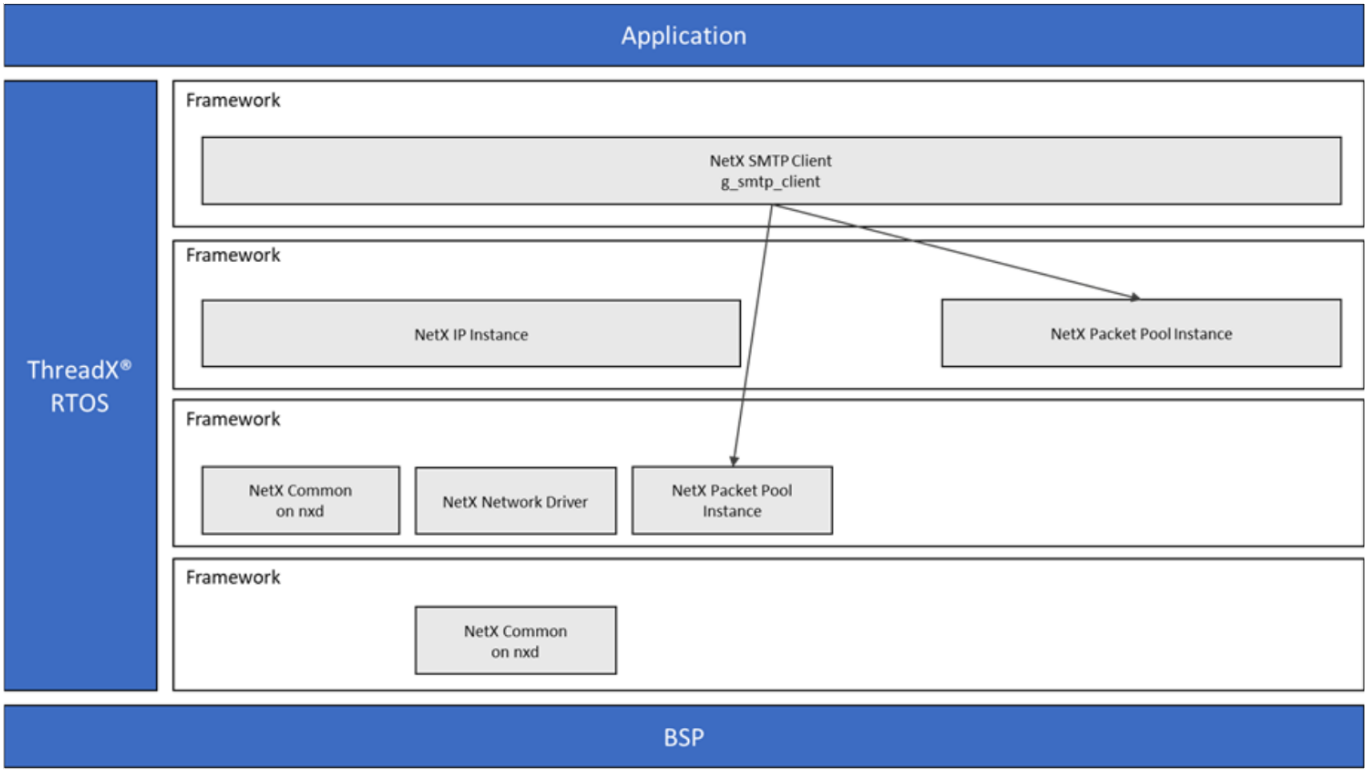


Figure 477: NetX/NetX Duo SMTP Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo SMTP Client Module in an Application for additional details.

4.3.25.2 NetX/NetX Duo SMTP Client Module APIs Overview

The NetX SMTP Client defines APIs for creating, deleting and sending mail. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo SMTP Client Module API Summary

Function Name	Example API Call and Description

nx_smtp_client_create	nxd_smtp_client_create(&demo_client, &client_ip, &client_packet_pool,USERNAME,PASSWORD, FROM_ADDRESS, LOCAL_DOMAIN, CLIENT_AUTHENTICATION_TYPE, server_ip_address, SERVER_PORT); Create an SMTP Client Instance.
**nxd_smtp_client_create	nxd_smtp_client_create(&demo_client, &client_ip, client_packet_pool, USERNAME, PASSWORD, from_address, client_domain, authentication_type, &server_address, port); Create an SMTP Client instance for IPv4 or IPv6 networks.
nx_smtp_client_delete	nx_smtp_client_delete(&demo_client); Delete an SMTP Client instance.
nx_smtp_mail_send	nx_smtp_mail_send(&demo_client, recipient_address, NX_SMTP_MAIL_PRIORITY_NORMAL, SUBJECT_LINE, MAIL_BODY, strlen(MAIL_BODY)); Create and send an SMTP Mail item.

Note

Fordetails on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo SMTP Client. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for definition of NetX Duo specific data types.

Status Return Values

Name	Description
NX_SUCCESS	SMTP call successful
NX_CALLER_ERROR*	Invalid caller of this service
NX_PTR_ERROR*	Invalid input pointer parameter
NX_SMTP_INVALID_PARAM*	Invalid non-pointer input
NX_IP_ADDRESS_ERROR*	Invalid IP address type
NX_SMTP_CLIENT_NOT_INITIALIZED	SMTP Client instance initialized for SMTP session

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.25.3 NetX/NetX Duo SMTP Client Module Operational Overview

A NetX IP instance is created and enabled for TCP services. The SMTP Client is created with a previously created packet pool as input, as well as information from the host, including the host's email address, mail domain, authentication type and server IP address** (Server IPv4 Address in the SMTP Client Module configuration table found below) and port (Server port in the SMTP Client Module configuration table found below). Server port defaults to the well-known SMTP port 25.

Note

nx_smtp_client_create is also available in NetX Duo POP3 Client; in this case, setting the Server IPv4 Address is correct. However, to use the SMTP Client with an IPv6 server, the application must supply an IPv6 address.

The payload of the packet pool used by the SMTP Client should be optimized to the typical SMTP data size plus network headers (IP, TCP and physical frame). For messages exceeding that packet payload, the SMTP Client will allocate additional packets for messages that exceed the packet payload. In the Packet Pool Instance configuration table found below, the setting for the packet pool payload defaults to 512 and the number of packets defaults to 16.

Note

The SMTP Client instance must have a fully qualified email address (Client Address in the SMTP Client Module configuration table found below). A fully qualified domain name contains a local-part and a domain name, separated by an '@' character. The domain name, which is usually the right half after the '@' symbol of the Client address, must also be specified (Client Domain in the configuration table below). This is used in the SMTP Client HELO and EHLO greeting to the Server.

NetX SMTP Authentication

The creation of the SMTP Client also requires setting Authentication type, Client Name and Client Password (In the SMTP Client Module configuration table found below). Client Names can either be fully qualified domain names or display user names.

Authentication is a way for SMTP Clients to prove their identity to the SMTP Server and have their mail delivered as trusted users. Most commercial SMTP Servers require that Clients be authenticated.

Typically, authentication data consists of the sender's username and password. During an authentication challenge, the Server prompts for this information and the Client responds by sending the requested data in encoded format. The Server decodes the data and attempts to find a match in its user database. If found, the Server indicates the authentication is successful.

There are two flavors of authentication: basic and digest. Digest is not supported in the current NetX SMTP Client, and will not be discussed here. Basic authentication is equivalent to the name and password authentication described above. In SMTP basic authentication, the name and passwords are base64 encoded. The advantage of basic authentication is its ease of implementation and widespread use. The main disadvantage of basic authentication is name and password data is transmitted openly in the request.

Plain Authentication

The NetX SMTP Client sends an AUTH command with the PLAIN parameter. The NetX SMTP Server will indicate if it supports this type of authentication. If so, the Client replies with a single base64 encoded username and password message to the Server. The Server responds with a status code indicating if the Client authentication is successful or not.

Login Authentication

The NetX SMTP Client sends an AUTH command with the LOGIN parameter. The SMTP Server will

indicate if it supports this type of authentication and start the authentication 'challenge'. It sends a base64 encoded prompt back to the Client which is typically "Username". The Client decodes the prompt, and replies with a base64 encoded username. If the Server accepts the Client username, it sends out a base64 encoded prompt for the Client password. The Client responds with a base64 encoded password. The Server will indicate if Client authentication is successful.

No Authentication

Some SMTP Servers are configured without authentication. If so, their 250 response to the Client EHLO message will not list any authentication types. However, no authentication types listed does not necessarily mean the Server does not require or support authentication. If the Client is configured for PLAIN or LOGIN authentication in this situation, the NetX Client thread task will default to PLAIN. If the Client is configured for NONE, the authentication step is skipped and the SMTP state advances to the MAIL state.

Note that if the Client is configured for no authentication and the SMTP Server does support authentication, the Client authentication type is switched to PLAIN.

Sending a Mail Message

After the SMTP Client is created, the SMTP Client application can start sending messages by calling the nx_smtp_mail_send service. Each time this service is called, NetX SMTP Client creates a new TCP connection with the SMTP server and begins an SMTP session. In this session, the Client sends a series of commands to the SMTP Server as part of the SMTP protocol, including passing authentication, and culminating in sending out the actual mail message. The TCP connection is then terminated, regardless of the outcome of the SMTP session. If the mail message is successfully sent, NX_SUCCESS is returned. If not the error code reflects either the SMTP Client error, e.g. failing authentication, or the underlying NetX error, e.g. failing to connect to the server.

After an SMTP session, regardless of success or failure in sending out a mail message, the SMTP Client is returned to the 'initial' state, and ready for another SMTP session (with the same SMTP server).

NetX/NetX Duo SMTP Client Module Important Operational Notes and Limitations

NetX/NetX Duo SMTP Client Module Operational Notes

- NetX SMTP Client API is compliant with RFC2821 "Simple Mail Transfer Protocol" and RFC 2554 "SMTP Service Extension for Authentication."

NetX/NetX Duo SMTP Client Module Limitations

- The NetX SMTP Client does not support CRAM-MD5 digest authentication.
- The NetX SMTP Client messages are limited to one recipient per mail item, and only one mail message per TCP connection with the SMTP server.
- SMTP commands VRFY, SEND, SOML, EXPN, SAML, ETRN, TURN and SIZE SMTP options are not supported.
- The SMTP Client is not mail browser ("mail user agent") which is typically used for creating the mail message. It is a "mail transfer agent" only. It will provide the necessary processing of the mail message body for SMTP transport as specified in RFC 2821. It does not check the contents for correct syntax e.g. the recipient and reverse pathway. There is no restriction what is in the mail buffer e.g. MIME data or clear text messages. Mail message format, specified in RFC 2822 for including headers and message body is beyond the scope of the SMTP Client API.
- Refer to the "NetX™ Simple Mail Transfer Protocol (SMTP) Client User Guide for the Renesas

Synergy™ Platform" for additional information on limitations.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.25.4 Including the NetX/NetX Duo SMTP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo SMTP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo SMTP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo SMTP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_smtp_client0 NetX SMTP Client	Threads	New Stack> X-Ware> NetX> Protocols> NetX SMTP Client
g_smtp_client0 NetX Duo SMTP Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo SMTP Client

When the NetX and/or NetX Duo SMTP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

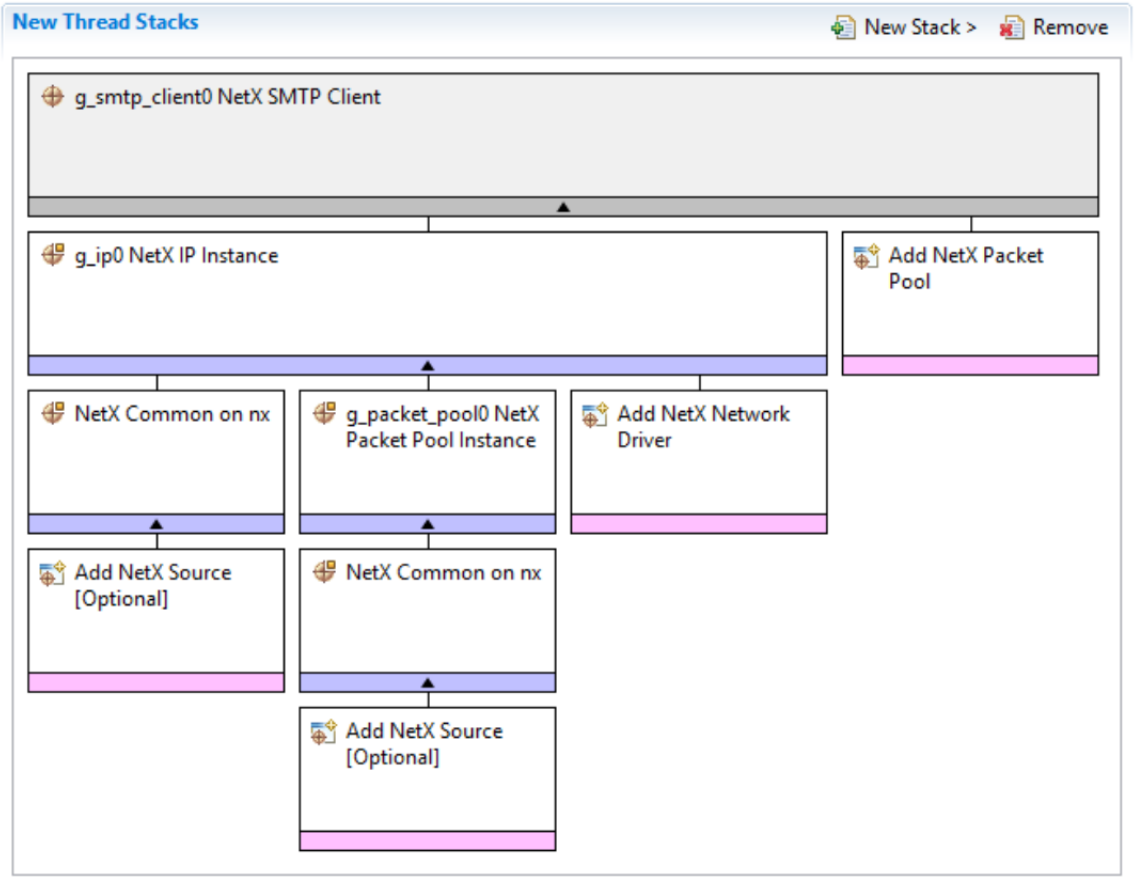


Figure 478: NetX/NetX Duo SMTP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.25.5 Configuring the NetX/NetX Duo SMTP Client Module

The NetX/NetX Duo SMTP Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo SMTP Client Module

ISDE Property	Value	Description
TCP window size (bytes)	1460	TCP window size selection
Packet allocation timeout (seconds)	2	Packet allocation timeout selection
TCP socket connect timeout (seconds)	10	TCP socket connect timeout selection
TCP socket disconnect timeout (seconds)	5	TCP socket disconnect timeout selection
Server greeting reply timeout	10	Server greeting reply timeout selection
Command timeout (seconds)	10	Command timeout selection
Mail data request timeout (seconds)	30	Mail data request timeout selection
TCP socket send completion timeout (seconds)	5	TCP socket send completion timeout selection
Server challenge maximum string length (bytes)	200	Server challenge maximum string length selection
Maximum password length (bytes)	20	Maximum password length selection
Maximum username length (bytes)	40	Maximum username length selection
Name	g_smtp_client0	Name selection
**Use server address type	IPv4, IPv6 Default: IPv6	Use server address type selection
Server IPv4 Address (use commas for separation)	192, 168, 0, 2	Server IPv4 Address selection
**Server IPv6 Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	Server IPv6 Address selection
Server Port	25	Server Port selection
Client Name	username	Client Name selection
Client Password	password	Client Password selection
Client Address	username@domain.com	Client Address selection
Client Domain	domain.com	Client Domain selection

Authentication Type	Login	Authentication Type selection
Name of generated initialization function	smtp_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different addresses for the Ethernet port. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo SMTP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection

ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo SMTP Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo SMTP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1

Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.25.6 Using the NetX/NetX Duo SMTP Client Module in an Application

The steps in using the NetX/NetX Duo SMTP Client module in a typical application are:

1. Create a mail message to send with nx_smtp_mail_send.
2. Continue sending mail messages as needed.
3. When done, delete the SMTP Client when done sending messages by calling nx_smtp_client_delete.

The following figure illustrates common steps in a typical operational flow diagram:

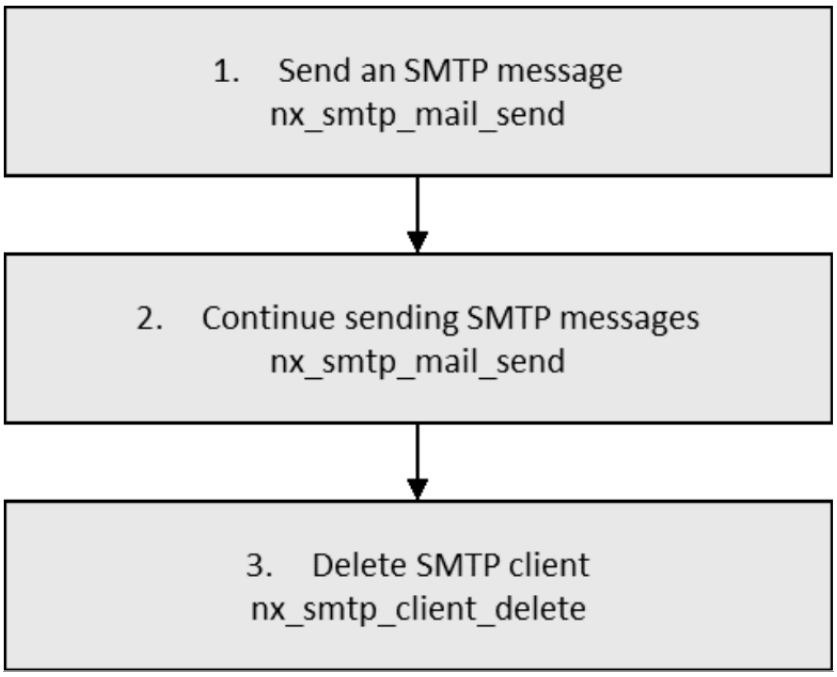


Figure 479: Flow Diagram of a Typical NetX/NetX Duo SMTP Client Module Application

4.3.26 NetX/NetX Duo SNMP Agent

4.3.26.1 NetX/NetX Duo SNMP Agent Introduction

The NetX/NetX Duo SNMP Agent module provides a high-level API for implementing the SNMP agent for the SSP architecture and is part of the NetX and NetX Duo application bundle included in the SSP Azure RTOS integration. This module is MCU independent, so any MCU that supports NetX and NetX Duo can implement the SNMP agent.

Note

Except for internal processing, the NetX Duo™ SNMP Agent module is identical in the application, set-up and running of an SNMP Agent session as the NetX™ SNMP Agent module.

NetX/NetX Duo SNMP Agent Module Features

- The NetX/NetX Duo SNMP Agent module is compliant with RFC1155, RFC1157, RFC1215, RFC1901, RFC1905, RFC1906, RFC1907, RFC1908, RFC2571, RFC2572, RFC2574, RFC2575, RFC 3414 and related RFCs.
- Operates only in UDP. TCP is not supported.
- Doesn't support Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS).
- The NetX/NetX Duo SNMP protocol implements SNMP Version 1, 2 and 3. The SNMPv3 implementation supports MD5 and Secure Hash Algorithm 1(SHA-1) authentication and Data Encryption Standard (DES) encryption. This version of the NetX and NetX Duo SNMP Agent has the following constraints:
    - One SNMP Agent per NetX IP Instance.
    - No support for RMON.

- SNMP v3 Inform messages are not supported
- Provides a mechanism to register callbacks for handling username, get, set and getnext when creating a SNMP agent.

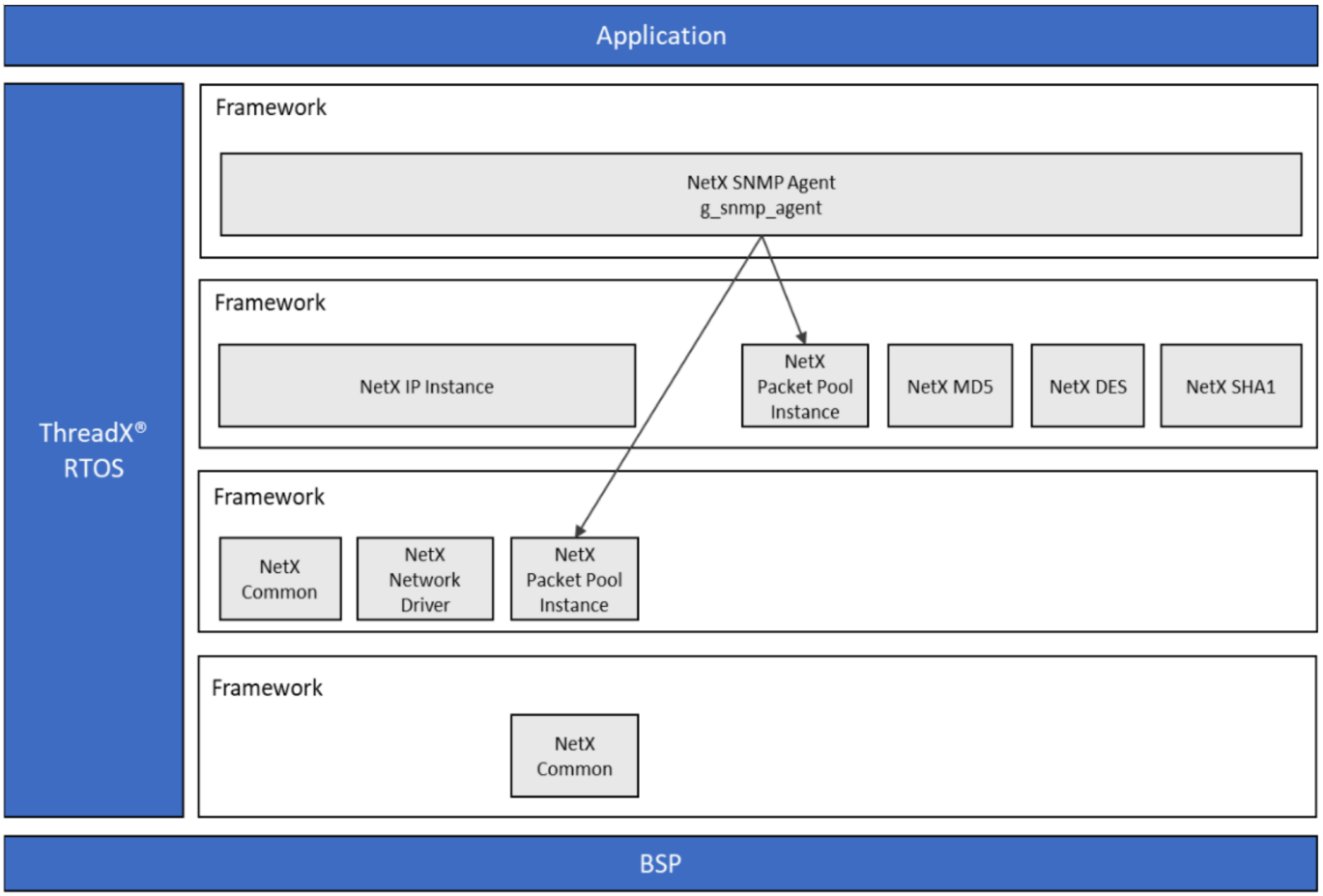


Figure 480: NetX/NetX Duo SNMP Agent Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo SNMP Agent Module in an Application for additional details.

4.3.26.2 NetX/NetX Duo SNMP Agent Module APIs Overview

The NetX Duo SNMP Agent defines APIs for creating and deleting the SNMP Agent instance and getting its messages. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo SNMP Agent Module API Summary (Part 1)

Function Name	Example API Call and Description

nx_snmp_agent_authenticate_key_use	(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key); Register a previously created authentication key with the SNMP Agent for SNMP Agent responses
nx_snmp_agent_auth_trap_key_use	(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key); Register a previously created authentication key with the SNMP Agent for SNMP trap messages.
nx_snmp_agent_community_get	(NX_SNMP_AGENT *agent_ptr, UCHAR **community_string_ptr); Retrieve the community string from the SNMP manager GET or GETNEXT request. The SNMP Agent should compare that to its own community string (see the nx_snmp_agent_public_string_test API description).
nx_snmp_agent_context_engine_set	(NX_SNMP_AGENT *agent_ptr, UCHAR *context_engine, UINT context_engine_size); Set the context engine ID of the SNMP Agent. Must specify the size of the ID string. Only used in SNMPv3 to identify the Agent to the SNMP Manager.
nx_snmp_agent_context_name_set	(NX_SNMP_AGENT *agent_ptr, UCHAR *context_name, UINT context_name_size); Set the context name of the SNMP Agent. This string must be null terminated and the size of the name must be specified. This name must be known to the SNMP Manager. Only used in SNMPv3.

nx_snmp_agent_create	(NX_SNMP_AGENT *agent_ptr, CHAR *snmp_agent_name, NX_IP *ip_ptr, VOID *stack_ptr, ULONG stack_size, NX_PACKET_POOL *pool_ptr, UINT (*snmp_agent_username_process)(struct NX_SNMP_AGENT_STRUCT *agent_ptr, UCHAR *username), UINT (*snmp_agent_get_process)(struct NX_SNMP_AGENT_STRUCT *agent_ptr, UCHAR *object_requested, NX_SNMP_OBJECT_DATA *object_data), UINT (*snmp_agent_getnext_process)(struct NX_SNMP_AGENT_STRUCT *agent_ptr, UCHAR *object_requested, NX_SNMP_OBJECT_DATA *object_data), UINT (*snmp_agent_set_process)(struct NX_SNMP_AGENT_STRUCT *agent_ptr, UCHAR *object_requested, NX_SNMP_OBJECT_DATA *object_data)); Create the SNMP Agent and set the Agent thread task size, packet pool for transmitting SNMP messages, and user defined callbacks for handling GET, GETNEXT, SET and username requests.
nx_snmp_agent_current_version_get	(NX_SNMP_AGENT *agent_ptr, UINT *version); Obtain the current version of SNMP based on the last message received.
nx_snmp_agent_delete	(NX_SNMP_AGENT *agent_ptr); Delete the SNMP instance
nx_snmp_agent_md5_key_create	(NX_SNMP_AGENT *agent_ptr, UCHAR *password, NX_SNMP_SECURITY_KEY *destination_key); Create a key based on a supplied password and SNMP Agent context engine ID using the MD5 algorithm. This key can be used for authentication or encryption.
nx_snmp_agent_privacy_key_use	(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key); Register a previously created security key with the SNMP Agent for encrypting and decrypting SNMPv3 messages.
nx_snmp_agent_priv_trap_key_use	(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key); Register a previously created security key with the SNMP Agent for encrypting SNMPv3 trap messages.
nx_snmp_agent_private_string_set	(NX_SNMP_AGENT *agent_ptr, UCHAR *private_string); Register a null terminated privacy string to the SNMP Agent. Only used in SNMP1 and SNMPv2.

nx_snmp_agent_private_string_test	(NX_SNMP_AGENT *agent_ptr, UCHAR *community_string, UINT *is_private); The SNMP Agent compares the privacy string in a SET request with the its own privacy string to determine if the SET request will be permitted.
nx_snmp_agent_public_string_set	(NX_SNMP_AGENT *agent_ptr, UCHAR *public_string); Register a null terminated public string to the SNMP Agent. Only used in SNMP1 and SNMPv2.
nx_snmp_agent_public_string_test	(NX_SNMP_AGENT *agent_ptr, UCHAR *community_string, UINT *is_public); The SNMP Agent compares the public string in a GET or GETNEXT request with the its own public string to determine if the request will be permitted.
nx_snmp_agent_request_get_type_test	(NX_SNMP_AGENT *agent_ptr, UINT *is_get_type); Determine if the last SNMP packet received was a GET, GETNEXT, or GET_BULT_REQUEST request type (returns TRUE) or a SET request (returns FALSE). Intended for use in the username callback for type of request received and checking public or private strings in the request message.
nx_snmp_agent_set_interface	(NX_SNMP_AGENT *agent_ptr, UINT if_index); Determine on which network interface to run the SNMP Agent protocol. The default interface is the primary (0) interface.
nx_snmp_agent_sha_key_create	(NX_SNMP_AGENT *agent_ptr, UCHAR *password, NX_SNMP_SECURITY_KEY *destination_key); Create a key based on a supplied password and SNMP Agent context engine ID using the SHA1 algorithm. This key can be used for authentication or encryption.
nx_snmp_agent_start	(NX_SNMP_AGENT *agent_ptr); Start the SNMP Agent thread task. This task waits to receive SNMP messages and formulates the response to the SNMP Manager.
nx_snmp_agent_stop	(NX_SNMP_AGENT *agent_ptr); Stop the SNMP Agent task thread. The SNMP Agent thread can be restarted by calling the nx_snmp_agent_start API.

nx_snmp_agent_trap_send	(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * username, UCHAR *enterprise, UINT trap_type, UINT trap_code, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv1. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed.
nx_snmp_agent_trapv2_send	(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR *username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv2. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed.
nx_snmp_agent_trapv3_send	(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv3. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed. Both the SNMP agent and browser must previously agree on the security (authentication and encryption) settings.
nx_snmp_agent_trapv2_oid_send	(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR *username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv2. This differs from nx_snmp_agent_trapv2_send in that it allows the caller to specify the OID directly.
nx_snmp_agent_trapv3_oid_send	(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv3. This differs from nx_snmp_agent_trapv3_send in that it allows the caller to specify the OID directly.
nx_snmp_agent_v3_context_boots_set	(NX_SNMP_AGENT *agent_ptr, UINT boots); Set the number of times the SNMP Agent has rebooted since the last communication with the SNMP Manager. Used in SNMPv3 only.

nx_snmp_agent_version_set	(NX_SNMP_AGENT *agent_ptr, UINT enabled_v1, UINT enable_v2, UINT enable_v3); Determine which type of SNMP packets the SNMP Agent will process. The application can choose V1, V2 and/or V3. Packets received from which the SNMP Agent is not enabled are dropped.
**nxd_snmp_agent_trap_send	(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR * username, UCHAR *enterprise, UINT trap_type, UINT trap_code, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv1 over IPv6. Note that this takes an NXD_ADDRESS* ip_address instead of ULONG ip_address.
**nxd _snmp_agent_trapv2_send	(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR * username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv2. Note that this takes an NXD_ADDRESS* ip_address instead of ULONG ip_address
**nxd _snmp_agent_trapv3_send	(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR *username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv3. Note that this takes an NXD_ADDRESS* ip_address instead of ULONG ip_address
nx_snmp_agent_trapv2_oid_send	(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address,, UCHAR *username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv2. Note that this takes an NXD_ADDRESS* ip_address instead of ULONG ip_address
**nxd _snmp_agent_trapv3_oid_send	(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR * username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr); Send a trap message in SNMPv3. Note that this takes an NXD_ADDRESS* ip_address instead of ULONG ip_address

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo SNMP Agent. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for definition of NetX Duo specific data types.

The following API calls are for processing data items into the SNMP Agent response.

NetX/NetX Duo SNMP Agent Module API Summary (Part 2)

Function Name	Example API Call and Description
nx_snmp_object_copy	(UCHAR *source_object_name, UCHAR *destination_object_name); This copies the string pointed to by source_object_name into the destination_object_name buffer (typically a trap message).
nx_snmp_object_counter_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This extracts the counter data from the location pointed to by source_ptr into the object data. Also used internally to copy internal counters of SNMPv3 statistics into error messages in SNMPv3 messages
nx_snmp_object_counter_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This sets the value of data extracted from the object_data into the location pointed to by the destination_ptr.
nx_snmp_object_counter64_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This extracts the counter data from the location pointed to by source_ptr into the object data. The difference with nx_snmp_objext_counter_get is the value is two words instead of one.
nx_snmp_object_counter64_set	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This sets the value of data extracted from the object_data into the location pointed to by the destination_ptr. The difference with nx_snmp_objext_counter_set is the value is two words instead of one.
nx_snmp_object_compare	(UCHAR *requested_object, UCHAR *reference_object); This compares the two input objects and if equal returns NX_SUCCESS.
nx_snmp_object_copy	(UCHAR *source_object_name, UCHAR *destination_object_name) This copies the data pointed to by the source object pointer into memory pointed to by the destination object pointer. Also used internally to copy internal counters of SNMPv3 statistics into error messages in SNMPv3 messages.

nx_snmp_object_end_of_mib	(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data); This appends an END_OF_MIB_VIEW macro as the input object's value. This signals the end of the MIB. See the snmp_mib_helper.h for an example.
nx_snmp_object_gauge_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This sets the input object to type SNMP GAUGE and places the value pointed to by the source_ptr into the object value.
nx_snmp_object_gauge_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This verifies the input object is an SNMP GAUGE data type, and extracts the value into the location pointed to by the destination pointer.
nx_snmp_object_id_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This function retrieves the object ID from the specified source location and writes it into the object data value.
nx_snmp_object_id_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This function retrieves the ASCII string from the input object and writes it to the area pointed to by the destination pointer.
nx_snmp_object_integer_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the object integer from the specified source location and stores it to the object data.
nx_snmp_object_integer_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the integer from the input object and places it in the destination.
nx_snmp_object_ip_address_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the IP address from the specified source location and stores it to the object data.
nx_snmp_object_ip_address_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the IP address from the input object and places it in the destination.
**nx_snmp_object_ipv6_address_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the IPv6 address from the specified source location and stores it to the object data.

**nx_snmp_object_ipv6_address_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the IPv6 address from the input object and places it in the destination.
nx_snmp_object_octet_string_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data, UINT length); This retrieves the string data from the specified source location and stores it to the object data.
nx_snmp_object_octet_string_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the string from the input object and places it in the destination.
nx_snmp_object_no_instance	(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data); This function places a no-instance value (NX_SNMP_ANS1_NO_SUCH_INSTANCE) in the object data.
nx_snmp_object_not_found	(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data); This function places a not-found value (NX_SNMP_ANS1_NO_SUCH_OBJECT) in the object data.
nx_snmp_object_string_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data, UINT length); This function retrieves the ASCII string from the specified source location and stores it to the object data.
nx_snmp_object_string_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the ASCII string from the input object and stores it to the destination.
nx_snmp_object_timetics_get	(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data); This function retrieves the data of type timer ticks from the specified source location and stores it to the object data.
nx_snmp_object_timetics_set	(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data); This retrieves the timer tick from the input object and stores it to the destination.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo SNMP Agent. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for definition of NetX Duo specific data types.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_PTR_ERROR*	Invalid input pointer parameter
NX_SNMP_UNSUPPORTED_AUTHENTICATION*	The authentication key is of an unknown or unsupported type (for example, not MD5 or SHA).
NX_SNMP_INVALID_PDU_ENCRYPTION*	The encryption key is of an unknown or unsupported type (for example, not MD5 or SHA).
NX_IP_ADDRESS_ERROR*	IP address supplied in a trap send API is null or invalid.
NX_SNMP_ERROR	Internal processing error e.g. not able to append data into the SNMP response.
NX_NOT_ENABLED	SNMP Agent is not enabled with the correct security settings for certain operations such as sending messages or processing authentication and encryption keys.
NX_SNMP_ERROR_TOOBIG	Data exceeds the size of the response buffer or exceed the allowable size of the parameter e.g. NX_SNMP_MAX_USER_NAME.
**NX_SNMP_INVALID_IP_PROTOCOL_ERROR	An IPv6 address is received in a trap send API but the NetX Duo library is not enabled for IPv6.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.26.3 NetX/NetX Duo SNMP Agent Module Operational Overview

The SNMP agent module makes use of the underlying NetX/NetX Duo stack to perform operations. Along with the IP stack, it makes use of the NetX MD5, NetX SHA1 and NetX DES modules for authentication and encryption in SNMP v3 operation.

The SNMP agent module can be created with the nx_snmp_agent_create API. For the implementation of the SNMP agent, the user needs to define the handler functions for username, get, getnext, and set operations.

NetX/NetX Duo SNMP Agent Module Important Operational Notes and Limitations

NetX/NetX Duo SNMP Agent Module Operational Notes

Disabling SNMP Version 1

The SNMP Agent module can disable processing of version 1 requests by selecting **Disable** in the

configuration properties for the SNMP Agent for **SNMP Version 1**. By default, this is enabled. When disabled, the SNMP Agent simply drops the packet with version 1, resulting in a timeout for the SNMP manager.

Disabling SNMP Version 2

The SNMP Agent module can disable processing of version 2 requests by selecting **Disable** in the configuration properties for the SNMP Agent for **SNMP Version 2**. By default, this is enabled. When disabled, the SNMP Agent simply drops the packet with version 2, resulting in a timeout for the SNMP manager.

Disabling SNMP Version 3

The SNMP Agent can disable processing of version 3 requests by selecting **Disable** in the configuration properties for the SNMP Agent for **SNMP Version 3**. By default, this is enabled. When disabled, the SNMP Agent simply drops the packet with version 3 resulting in a timeout for the SNMP manager. Enabling SNMP version 3 requires MD5, SHA1 authentication and DES encryption.

NetX/NetX Duo SNMP Agent Module Limitations

- One SNMP Agent per NetX IP Instance.
- No support for RMON
- SNMP v3 Informs are not supported
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.26.4 Including the NetX/NetX Duo SNMP Agent Module in an Application

This section describes how to include either or both the NetX and NetX Duo SNMP Agent module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo SNMP Agent module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo SNMP Agent Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_snmp_agent0 NetX SNMP Agent	Threads	New Stack> X-Ware> NetX> Protocols> NetX SNMP Agent
g_snmp_agent0 NetX Duo SNMP Agent	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo SNMP Agent

When the NetX and/or NetX Duo SNMP Agent module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink

band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

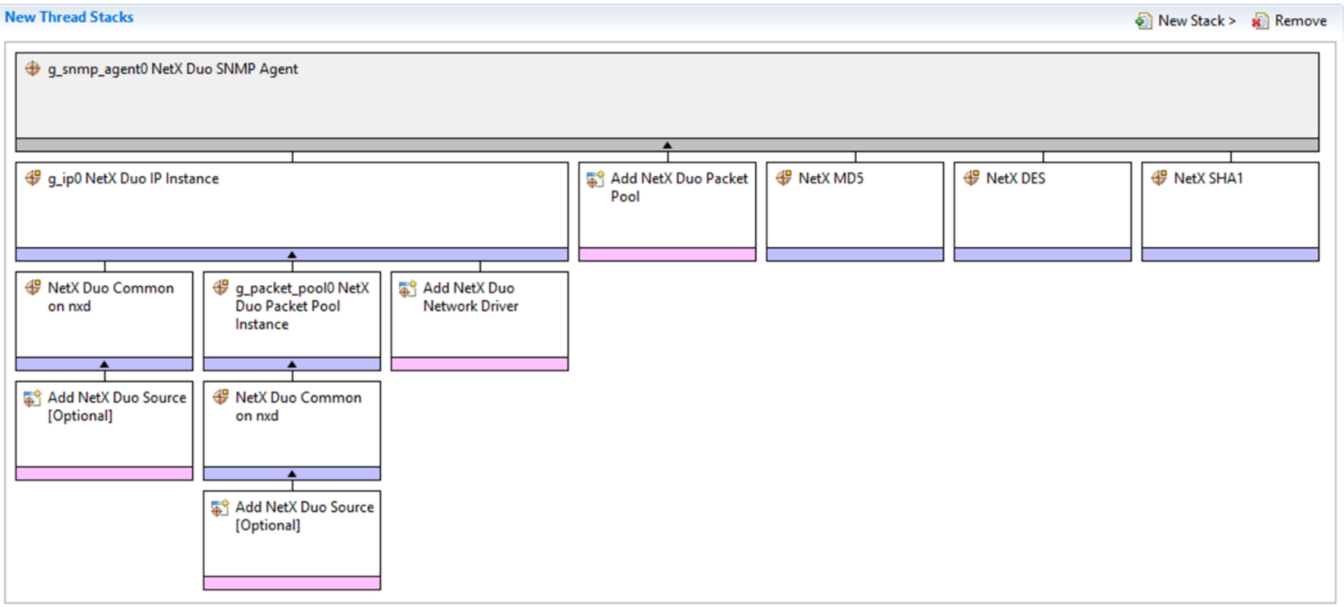


Figure 481: NetX/NetX Duo SNMP Agent Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.26.5 Configuring the NetX/NetX Duo SNMP Agent Module

The NetX/NetX Duo SNMP Agent module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to

learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo SNMP Agent Module

ISDE Property	Value	Description
Internal thread stack size (bytes)	4096	Internal thread stack size selection
SNMP agent priority	16	SNMP agent priority selection
Type of service for SNMP responses	Normal, Minimum Delay, Maximum Delay, Maximum Reliability, Minimum Cost Default: Normal	Type of service for SNMP responses selection
Fragment enable for SNMP PDU requests	Fragment, Don't Fragment Default: Don't Fragment	Fragment enable for SNMP PDU requests selection
SNMP socket time to live	128	SNMP socket time to live selection
Agent timeout	100	Agent timeout selection
Max octet string size	255	Max octet string size selection
Max content string size	32	Max content string size selection
Max User Name Size	64	Max user name size selection
Max security Key Size	64	Max security key size selection
Minimum SNMP packet size	560	Minimum SNMP packet size selection (Value must be between 560 to 1500)
UDP port number	161	UDP port number selection (Value must be between 1 to 65535)
Trap destination port	162	Trap destination port selection
Max trap Name Size	64	Max trap Name Size selection
Max trap Key Size	64	Max trap Key Size selection
Interval between SNMP packet processing timer ticks	100	Interval between SNMP packet processing timer ticks selection
SNMP Version 1	Enable, Disable Default: Enable	SNMP Version 1 selection
SNMP Version 2	Enable, Disable Default: Enable	SNMP Version 2 selection

SNMP Version 3	Enable, Disable Default: Enable	SNMP Version 3 selection
Name	g_snmp_agent0	Name selection
Read Community String	public	Read Community String selection (Must be less than 64 chars)
Write Community String	private	Write Community String selection (Must be less than 64 chars)
Name of SNMP Username Handler	sf_snmp0_username_handler	Name of SNMP Username Handler selection
Name of SNMP GET Handler	sf_snmp0_get_handler	Name of SNMP GET Handler selection
Name of SNMP GETNEXT Handler	sf_snmp0_getnext_handler	Name of SNMP GETNEXT Handler selection
Name of SNMP SET Handler	sf_snmp0_set_handler	Name of SNMP SET Handler selection
Name of generated initialization function	snmp_agent_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto Initialization selection
SNMP agent instance id	0	SNMP agent instance id selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different addresses for the Ethernet port. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo SNMP Agent Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo MD5 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo DES Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo SHA1 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo SNMP Agent Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo SNMP Agent Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin

TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.26.6 Using the NetX/NetX Duo SNMP Agent Module in an Application

The steps in using the NetX/NetX Duo SNMP Agent module in a typical application are:

1. Create an agent with nx_snmp_agent_create API
2. Define handler functions
    a. username handler function nx_snmp_agent_username_process in user code
    b. get handler function nx_snmp_agent_get_process in user code
    c. get next handler function nx_snmp_agent_getnext_process in user code
    d. set handler function nx_snmp_agent_set_process in user code
3. Create a structure to map OIDs to action handlers in user code
4. Configure read, write community strings nx_snmp_agent_public_string_set and nx_snmp_agent_private_string_set
5. If the user wants to use SNMPv3, they must create the keys using the nx_snmp_agent_md5_key_create API or the nx_snmp_agent_sha_key_create and associate the keys with authentication, encryption and trap services nx_snmp_agent_authenticate_key_use, nx_snmp_agent_privacy_key_use, nx_snmp_agent_auth_trap_key_use and nx_snmp_agent_priv_trap_key_use
6. Start the SNMP agent using the nx_snmp_agent_start API

The following figure illustrates common steps in a typical operational flow diagram:

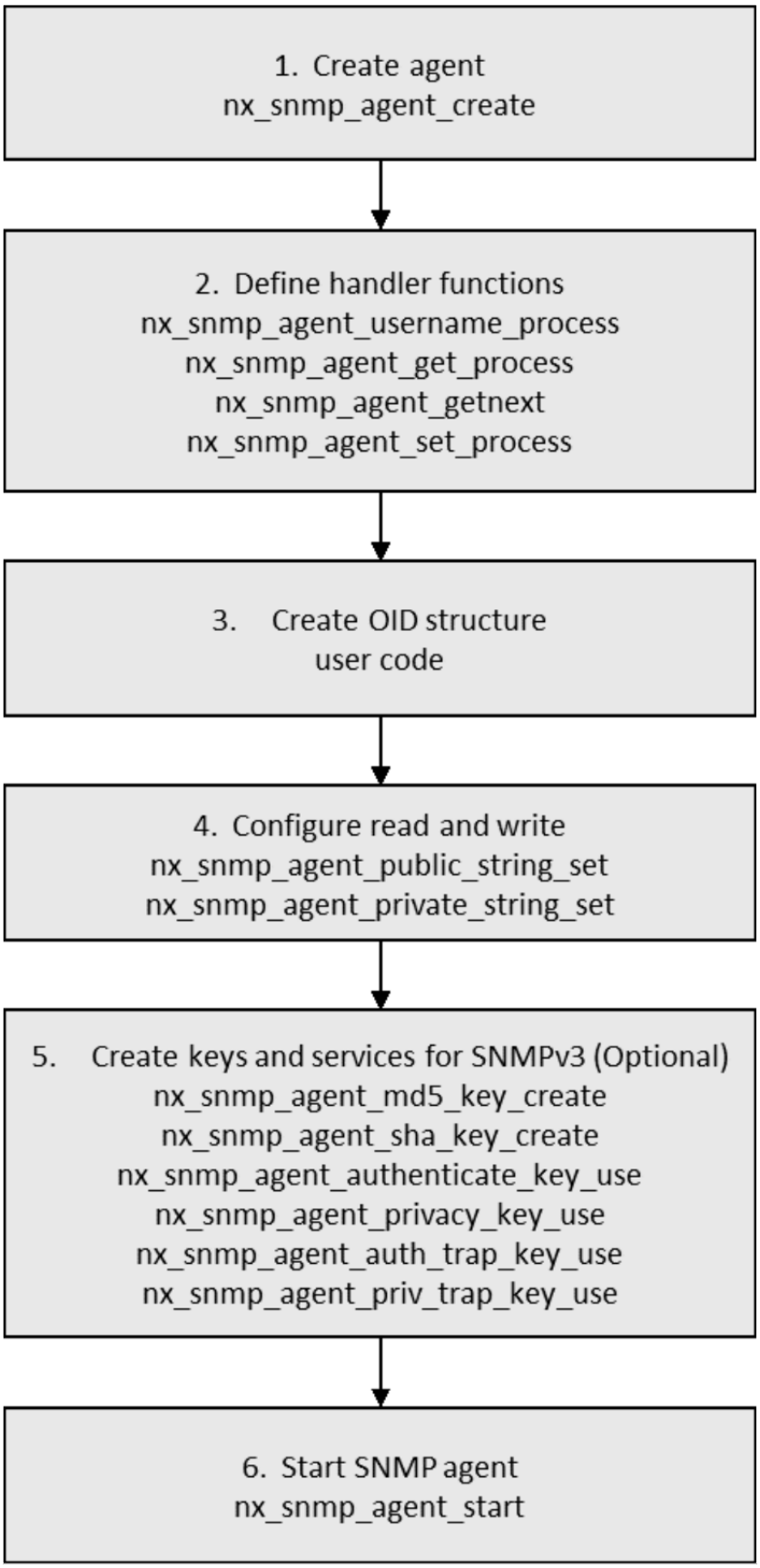


Figure 482: Flow Diagram of a Typical NetX/NetX Duo SNMP Agent Module Application

4.3.27 NetX/NetX Duo SNTP Client

4.3.27.1 NetX/NetX Duo SNTP Client Introduction

The Simple Network Time Protocol (SNTP) is a protocol designed for synchronizing clocks over the Internet. SNTP Version 4 is a simplified protocol based on the Network Time Protocol (NTP) and utilizes User Datagram Protocol (UDP) services to perform time updates in a simple, stateless protocol. On the internet, the SNTP provides accuracies of 1-50 milliseconds, depending on the characteristics of the synchronization source and network paths. The SNTP has many options to provide reliability of receiving time updates. Ability to switch to alternative servers, applying back-off polling algorithms and automatic time server discovery are just a few of the means for an SNTP Client to handle a variable internet time service environment. What it lacks in precision it makes up for in simplicity and ease of implementation. The SNTP is intended primarily for providing comprehensive mechanisms to access national time and frequency dissemination for NTP Server services.

Note

Except for internal processing, the NetX Duo™ SNTP Client is identical in the application, set up and running of an SNTP thread as the NetX™ SNTP Client. This document will clearly identify any differences in use between the NetX and NetX Duo SNTP Client.

Unsupported Features

Broadcast has not been tested in this version of SSP.

Multiple network interface has not been tested in this version of SSP.

NetX/NetX Duo SNTP Client Module Features

- Supports the SNTP standard to obtain time over the internet
- The NetX SNTP client is compliant with RFC4330 Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI and related RFCs
- Supports both unicast and broadcast SNTP messaging
- Utilities for converting NTP time into date and time format.
- Performs sanity checks for valid time reception
- Support for a SNTP Client on secondary interfaces
- Callbacks to be notified of receiving time updates

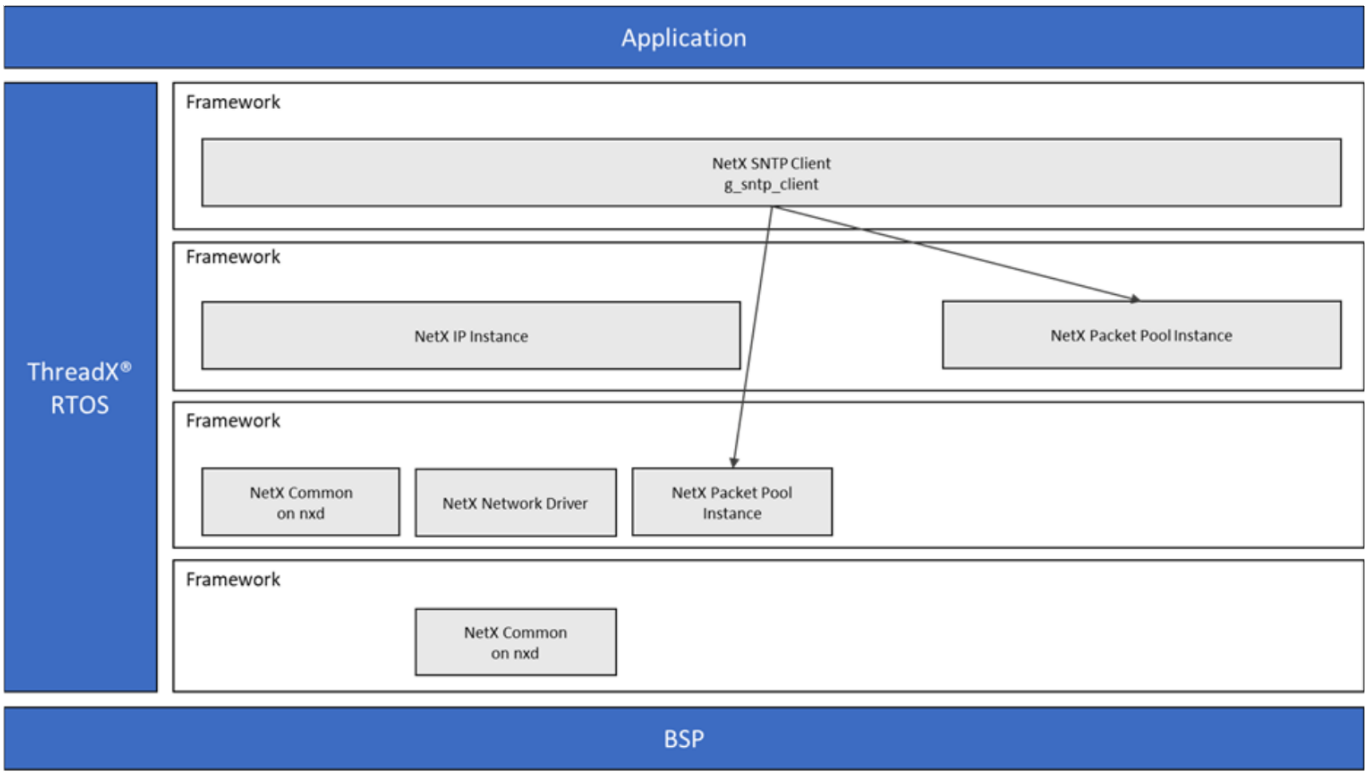


Figure 483: NetX/NetX Duo SNTP Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo SNTP Client Module in an Application for additional details.

4.3.27.2 NetX/NetX Duo SNTP Client Module APIs Overview

The NetX SNTP Client module defines APIs for creating, deleting, generating response packets, response sending and getting information from a received packet. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo SNTP Client Module API Summary

Function Name	Example API Call and Description
nx_sntp_client_create	nx_sntp_client_create(&demo_client, iface_index,&client_ip, &client_packet_pool, leap_second_handler, kiss_of_death_handler,NULL // no random_number_generator callback); This service creates an SNTP Client instance.
nx_sntp_client_delete	nx_sntp_client_delete(&demo_client); Delete the SNTP Client.
nx_sntp_client_get_local_time	nx_sntp_client_get_local_time(&demo_client, &base_seconds,&base_milliseconds, NX_NULL); Get SNTP Client local time.

nx_sntp_client_initialize_broadcast	nx_sntp_client_initialize_broadcast(&demo_client ,0x0, NX_NULL, IP_ADDRESS(192,2,2,255); Initialize Client for IPv4 broadcast operation.
**nxd_sntp_client_initialize_broadcast	nxd_sntp_client_initialize_broadcast(&demo_clie nt,0x0, NX_NULL, &broadcast_server) Initialize Client for IPv6 or IPv4 broadcast operation.
nx_sntp_client_initialize_unicast	nx_sntp_client_initialize_unicast(&demo_client, IP_ADDRESS(192,2,2,1)); Initialize Client for IPv4 unicast operation.
**nxd_sntp_client_initialize_unicast	nxd_sntp_client_initialize_unicast(&demo_client, *unicast_server); Initialize Client for IPv4 or IPv6 unicast operation.
nx_sntp_client_receiving_updates	nx_sntp_client_receiving_updates(&demo_client, &receive_status); Client is currently receiving valid SNTP updates.
nx_sntp_client_request_unicast_time	nx_sntp_client_request_unicast_time(&demo_clie nt, 400); Send a request asynchronously to NTP server.
nx_sntp_client_run_broadcast	nx_sntp_client_run_broadcast(&demo_client); Receive time updates from server.
nx_sntp_client_run_unicast	nx_sntp_client_run_unicast(&demo_client); Send requests and receive time updates from server.
nx_sntp_client_set_local_time	nx_sntp_client_set_local_time(&demo_client, base_seconds, base_fraction); Set SNTP Client initial local time.
nx_sntp_client_set_time_update_notify	nx_sntp_client_set_time_update_notify(&demo_cl ient, time_update_cb); Sets callback to notify the application when the SNTP Client receives a valid time update.
nx_sntp_client_stop	nx_sntp_client_stop(&demo_client); Stop the SNTP Client thread.
nx_sntp_client_utility_display_date_and_time	nx_sntp_client_utility_display_date_time(&demo_ client, buffer, sizeof(buffer)); Display NTP time in date and time format
nx_sntp_client_utililty_msecs_to_fraction	nx_sntp_client_utility_msecs_to_fraction(millisec onds, &fraction); Convert milliseconds to NTP fraction component.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo SNTP Client. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for definition of NetX Duo specific data types.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_SNTP_INSUFFICIENT_PACKET_PAYLOAD	Invalid non-pointer input
NX_PTR_ERROR **	Invalid pointer input
NX_CALLER_ERROR**	Invalid caller of service
NX_INVALID_INTERFACE	Invalid network interface
NX_INVALID_PARAMETERS	Invalid non-pointer input
NX_SNTP_PARAM_ERROR	Invalid non-pointer input
NX_SNTP_CLIENT_NOT_STARTED	SNTP Client not running
NX_SNTP_CLIENT_ALREADY_STARTED	Client already running
NX_SNTP_CLIENT_NOT_INITIALIZED	Client not initialized
NX_SNTP_ERROR_CONVERTING_DATETIME	NX_SNTP_CURRENT_YEAR not defined or no local time established
NX_SNTP_INVALID_DATETIME_BUFFER	Insufficient buffer length
NX_SNTP_OVERFLOW_ERROR	Error converting time to a date
NX_SNTP_INVALID_TIME	Invalid SNTP data input

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.27.3 NetX/NetX Duo SNTP Client Module Operational Overview

The NetX/NetX Duo SNTP Client module creates an IP instance and packet pool. In addition, the UDP is enabled on that IP instance and a UDP socket is created and bound to the well-known port 123 for sending time update requests to and receiving time data from an SNTP Server, although alternative ports will work as well. Broadcast clients should bind the UDP port when their broadcast server is sending on, usually 123. The NetX/NetX Duo SNTP Client application requires a server address to initialize a SNTP Client thread task for either unicast or broadcast operations.

The NetX/NetX Duo SNTP Client can operate in one of two basic modes, unicast or broadcast, to obtain time over the internet. In unicast mode, the Client polls its SNTP Server on regular intervals and waits to receive a reply from that server. When one is received, the client verifies that the reply contains a valid time update by applying a set of sanity checks recommended by RFC 4330. The client then applies the time difference locally, if there is a time difference between the server clock and the local clock. In broadcast mode, the client merely listens for time update broadcasts and maintains its local clock after applying a similar set of sanity checks to verify the update time data.

Before the client can run in either mode, the application can get an estimate of the absolute time. In NTP, that means the number of seconds since Jan 1, 1970 at midnight. For example:

```
    base_seconds = 0xd2c96b90; /* This is the number of seconds since
1970
                                   on Jan 24, 2012 UTC */
    base_fraction = 0xa132db1e; /* between 0-1 seconds, not so critical */
```

It is not required, but if this time is registered with the SNTP Client (using the nx_sntp_client_set_local_time service), it gives the SNTP Client a means to gauge if the first received time update is reasonably accurate. NTP time keepers are more accurate than all but the most demanding time-based applications, but there is the need to check for invalid or rogue SNTP packets. If obtaining this baseline time is not possible, and no local time is set before starting the SNTP Client, the SNTP Client will accept the time updates at face value.

Now the application can prepare the SNTP Client for receiving time updates. This is done by calling either nx_sntp_client_initialize_unicast, or nx_sntp_client_initialize_broadcast for unicast or broadcast modes. (These are limited to IPv4 networks only.) If using the NetX Duo SNTP Client Module, nxd_sntp_client_initialize_broadcast and nxd_sntp_client_initialize_unicast are available services which support both IPv4 and IPv6 networks. If these initialization service calls succeed, the application can start the SNTP Client by calling the nx_sntp_client_run_broadcast or nx_sntp_client_run_unicast services.

The application can set the following parameters to fine-tune SNTP Client operation:

- The maximum time adjustment to make the local clock, which is the Maximum time adjustment allowed to local clock time (milliseconds) property. If there is no notion of local time, the Ignore maximum time adjust limit at startup property can be enabled to skip this check on the first time update.
- The unicast polling rate which is the Starting poll interval for unicast update request (seconds) property. This is the rate at which the SNTP Client polls the NTP time server for an update if running in unicast mode.
- The maximum allowed interval between time requests sent to the server, which is the Maximum time lapse without valid update (seconds) property. If a time request goes unanswered, the SNTP Client doubles the polling interval up to this maximum interval.
- The maximum number of invalid SNTP packets received from a server before marking the server invalid, which is the Invalid message limit to mark server invalid property.

If these limits are exceeded, the SNTP Client continues to run but sets the current SNTP Server status to invalid. The application can call the nx_sntp_client_receiving_updates service periodically to check for valid server status. If this service indicates no updates received, the application should stop the SNTP Client thread using the nx_sntp_client_stop service and use another SNTP Server. To restart the SNTP Client, call either of the two initialize services, nx_sntp_client_initialize_broadcast or nx_sntp_client_initialize_unicast for IPv4 networks (or nxd_sntp_client_initialize_broadcast or nxd_sntp_client_initialize_unicast), then restart the SNTP Client with the nx_sntp_client_run_broadcast or nx_sntp_client_run_unicast services.

Additional details on the topics of local clock operation, SNTP sanity checks, and SNTP asynchronous and unicast requests are described in detail in the *NetX Simple Network Time Protocol (SNTP) Client User Guide* for the Renesas Synergy™ Platform document described in the introduction to this document.

NetX/NetX Duo SNTP Client Module Important Operational Notes and Limitations

NetX/NetX Duo SNTP Client Module Operational Notes

- The application can be notified of time updates if it registers a callback with the SNTP Client. To do so, use the _nx_sntp_client_set_time_update_notify service to specify the callback.
- Once the SNTP Client begins receiving time updates, the application can query the SNTP Client for status on time updates. It can receive the NTP time directly using the nx_sntp_client_get_local_time service, or it can receive a formatted date time output using the _nx_sntp_client_utility_display_date_time service. In the latter case, the application must set the Current calendar year property of the SNTP Client module for SNTP Client to process NTP time data correctly.
- RFC 4330 recommends that SNTP clients should operate only at the highest stratum of their local network and preferably in configurations where no NTP or SNTP Client is dependent on them for synchronization. Stratum level reflects the host position in the NTP time hierarchy where stratum 1 is the highest level (a root time server) and 15 is the lowest allowed level (per client). The SNTP Client default minimum stratum is 2.

NetX/NetX Duo SNTP Client Module Limitations

- Precision in local time representation in NTP time updates handled by the SNTP Client API is limited to millisecond resolution.
- The NetX/NetX Duo SNTP Client only holds a single SNTP Server address at any time. If that server appears to be no longer valid, the application must stop the SNTP Client task and reinitialize it with another SNTP server address, using either broadcast or unicast SNTP communication.
- The NetX/NetX Duo SNTP Client does not support many cast.
- The NetX/NetX Duo SNTP Client does not support authentication mechanisms for verifying received packet data.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.27.4 Including the NetX/NetX Duo SNTP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo SNTP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo SNTP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo SNTP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sntp_client0 NetX SNTP Client	Threads	New Stack> X-Ware> NetX> Protocols> NetX SNTP Client
g_sntp_client0 NetX Duo SNTP Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo SNTP Client

When the NetX and/or NetX Duo SNTP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

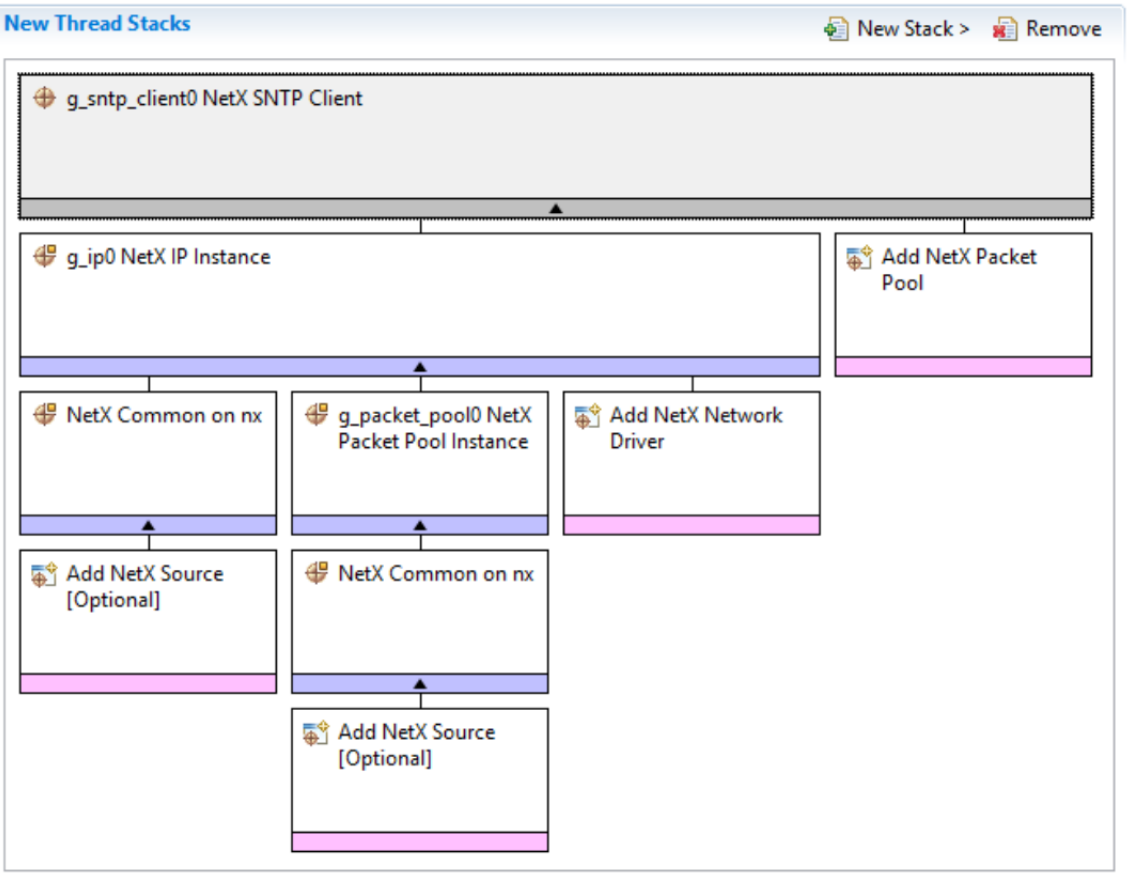


Figure 484: NetX/NetX Duo SNTP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.27.5 Configuring the NetX/NetX Duo SNTP Client Module

The NetX/NetX Duo SNTP Client module must be configured by the user for the desired operation.

The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo SNTP Client Module

ISDE Property	Value	Description
Internal thread stack size (bytes)	2048	Internal thread stack size selection
SNTP client thread time slicing interval (ticks)	TX_NO_TIME_SLICE	SNTP client thread time slicing interval selection
Internal thread priority	2	Internal thread priority selection
UDP socket name	SNTP Client socket	UDP socket name selection
UDP port number	123	UDP port number selection
Server UDP port	123	Server UPD port selection
Time to live	128	Time to live selection
Maximum UDP packets queue depth (units)	5	Maximum UDP packets queue depth selection
Packet allocation timeout (seconds)	1	Packet allocation timeout selection
SNTP version to use	3	SNTP version to use selection
NTP minimum version	3	NTP minimum version selection
Lowest level server stratum client accepts	2	Lowest level server stratum client accepts selection
Minimum time difference that triggers adjustment (milliseconds)	10	Minimum time difference that triggers adjustment selection
Maximum time adjustment allowed to local clock time (milliseconds)	180000	Maximum time adjustment allowed to local clock time selection
Ignore maximum time adjust limit at startup	True, False Default: True	Ignore maximum time adjust limit at startup selection

Maximum time lapse without valid update (seconds)	7200	Maximum time lapse without valid update selection
Update time remaining timer update interval (seconds)	1	Update time remaining timer update interval selection
Starting poll interval for unicast update request (seconds)	3600	Starting poll interval for unicast update request selection
Poll interval increment after failed time update	2	Poll interval increment after failed time update selection
Calculate round trip time of messages	True, False Default: False	Calculate round trip time of messages selection
**Maximum server clock inaccuracy to accept (to disable set to 0)	50000	Maximum server clock inaccuracy to accept selection
Invalid message limit to mark server invalid	3	Invalid message limit to mark server invalid selection
Randomize update request interval on startup	True, False Default: False	Randomize update request interval on startup selection
Internal Task sleep interval (ticks)	1	Internal task sleep interval selection
Current calendar year	2016	Current calendar year selection
Name	g_sntp_client0	Module name
Index to SNTP Network Interface	0	Index to SNTP network interface selection
Name of Leap Second Handler	leap_second_handler	Name of leap second handler selection
Name of Kiss of Death Handler	kiss_of_death_handler	Name of kiss of death handler selection
Name of Random Number Generator Function (optional)	NULL	Name of random number generator function selection
Name of generated initialization function	sntp_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it

might be useful to select different addresses for the Ethernet port. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo SNTP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection

ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo SNTP Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX/NetX Duo SNTP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin

MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.27.6 Using the NetX/NetX Duo SNTP Client Module in an Application

The steps in using the NetX/NetX Duo SNTP Client module in a typical application are:

1. Wait for a valid IP address using the nx_ip_status_check API.
2. Set the mode for access to the SNTP Server with nx_sntp_client_initialize_unicast or nx_sntp_client_initialize_broadcast API, or if using NetX Duo nxd_sntp_client_initialize_unicast or nxd_sntp_client_initialize_broadcast API.
3. Set the local time with the nx_sntp_client_set_local_time API (optional).
4. Run the client using the nx_sntp_client_run_unicast or nx_sntp_client_run_broadcast API.
5. Check if the SNTP is valid and running with the nx_sntp_client_receiving_updates API (recommended).
6. Get the local time with the nx_sntp_client_get_local_time API or get the local time in date time format with the nx_sntp_client_utility_display_date_time API (optional).
7. Stop the SNTP service with the nx_sntp_client_stop API.

The following figure illustrates common steps in a typical operational flow diagram:

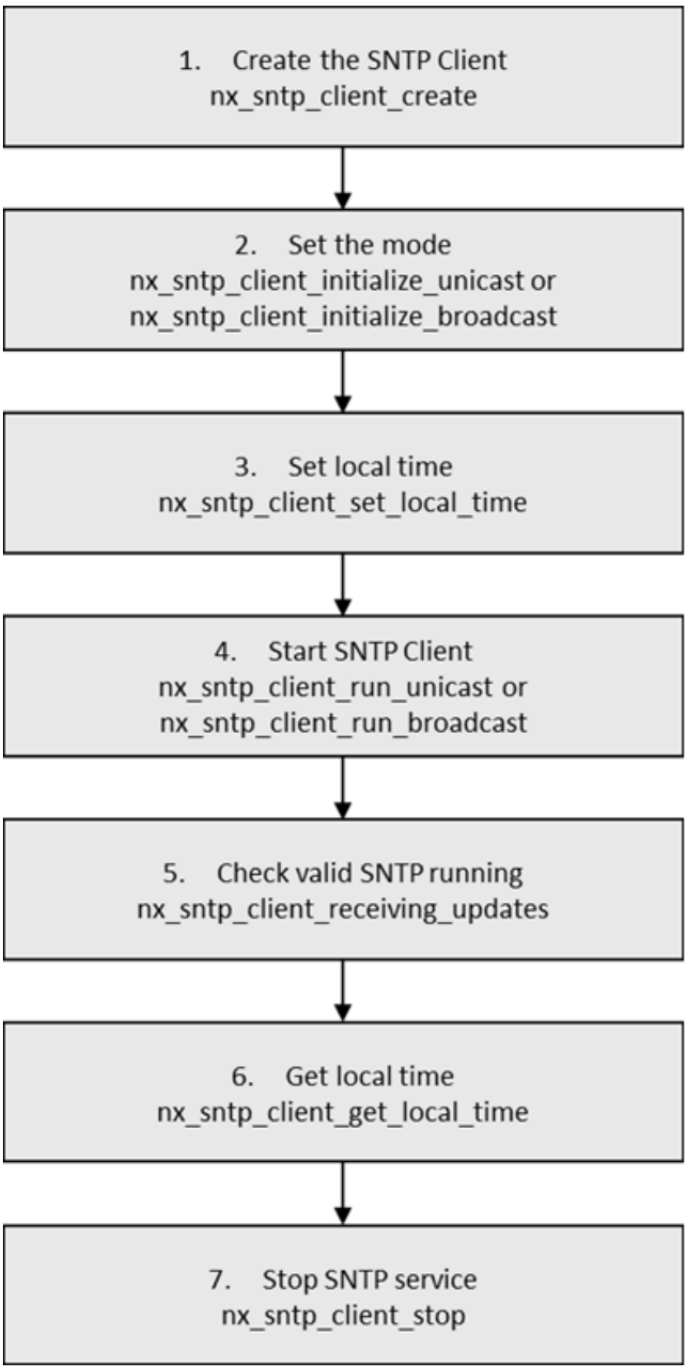


Figure 485: Flow Diagram of a Typical NetX/NetX Duo SNTP Client Module Application

4.3.28 NetX/NetX Duo POP3 Client

4.3.28.1 NetX/NetX Duo POP3 Client Introduction

The Post Office Protocol Version 3 (POP3) is a protocol designed to provide a mail transport system for small workstations to access Client maildrops on POP3 Servers. The POP3 utilizes Transmission

Control Protocol (TCP) services to perform mail transfers.

Client mail is stored on a POP3 Server in a mailbox or "maildrop." A maildrop is represented as a 1-based array of mail items where each mail is retrieved and deleted using its index in the maildrop. Once a mail message is downloaded, the Client typically marks the mail item for deletion in the Server's maildrop (the Server will probably do this automatically anyway). When finished retrieving mail items, the application should delete the POP3 Client instance and close the TCP connection. To retrieve the mail again, another POP3 Client must be created.

Note

Except where noted, the NetX Duo POP3 Client module is identical in the application, set up and running of a POP3 session as the NetX POP3 Client module. For setting up the IP instance for IPv6 in NetX Duo, please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform.

NetX/NetX Duo POP3 Client Module Features

- NetX Client POP3 is compliant with RFC 1939.
- Provides high-level APIs to:
  - Create and delete POP3 Client instances
  - Query for number and size of mail messages to receive
  - Receive individual mail messages
  - Remove mail messages from the Server maildrop box

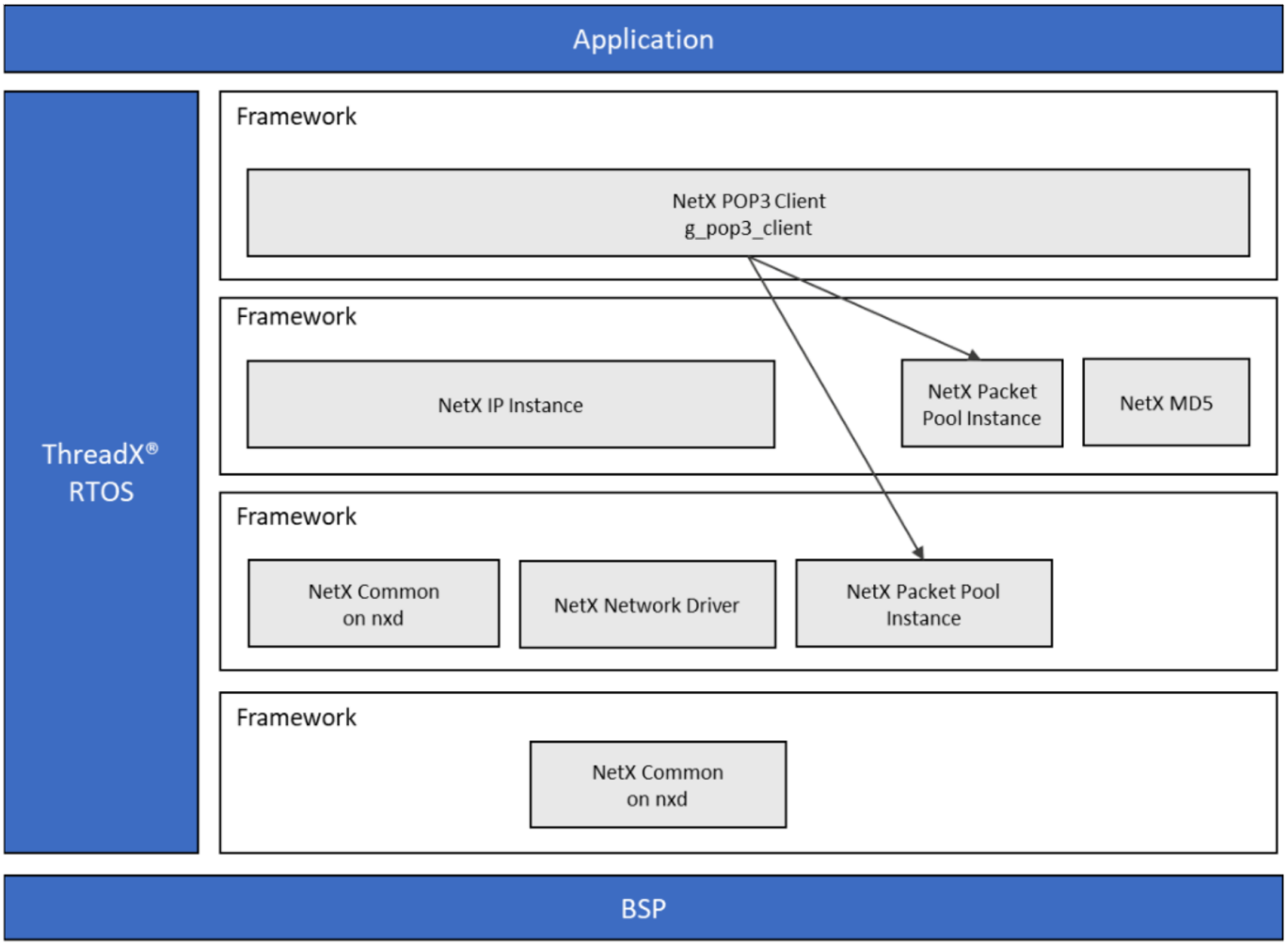


Figure 486: NetX/NetX Duo POP3 Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo POP3 Client Module in an Application for additional details.

4.3.28.2 NetX/NetX Duo POP3 Client Module APIs Overview

The NetX POP3 Client defines APIs for creating and deleting the POP3 instance and getting its messages. A complete list of the available APIs, an example API call and a short description of each can be found in the following table.

NetX/NetX Duo POP3 Client Module API Summary

Function Name	Example API Call and Description
nx_pop3_client_create	nx_pop3_client_create(&demo_client, NX_FALSE/* disable APOP authentication */, &client_ip, &client_packet_pool, POP3_SERVER_ADDRESS, POP3_SERVER_PORT, LOCALHOST, LOCALHOST_PASSWORD); Create a POP3 Client Instance for IPv4 only.
**nxd_pop3_client_create	nxd_pop3_client_create(&demo_client, NX_FALSE/* disable APOP authentication */, &client_ip, &client_packet_pool, *server_ip_address, ULONG server_port, CHAR *client_name, CHAR *client_password); Create a POP3 Client Instance for either IPv4 or IPv6
nx_pop3_client_delete	nx_pop3_client_delete (&demo_client); Delete a POP3 Client Instance
nx_pop3_client_mail_item_delete	nx_pop3_client_mail_item_delete(&demo_client, item_index); Delete a Client mail item for a Server maildrop.
nx_pop3_client_mail_item_get	nx_pop3_client_mail_item_get (&demo_client, 1, &item_size); Retrieve a specific mail message size.
nx_pop3_client_mail_items_get	nx_pop3_client_mail_item_get (&demo_client, 1, &number_mail_items, &maildrop_total_size); Obtain the number of mail items in a maildrop.
nx_pop3_client_mail_item_message_get	nx_pop3_client_mail_item_message_get (&demo_client, &recv_packet_ptr, &bytes_retrieved, &final_packet); Download a specific mail message.
nx_pop3_client_mail_item_size_get	nx_pop3_client_mail_item_size_get (&demo_client, mail_item, &size); Obtain the size of a specific mail item.
nx_pop3_client_quit	nx_pop3_client_quit(&demo_client); Sends a QUIT command to the POP3 server.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo POP3 Client. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_PTR_ERROR*	Invalid input pointer parameter
NX_POP3_CLIENT_INVALID_INDEX*	Null mail index input
NX_POP3_PARAM_ERROR	Invalid non-pointer input
NX_POP3_APOP_FAILED_MD5_DIGEST	POP3 Client failed APOP authentication
NX_POP3_INVALID_MAIL_ITEM	Invalid mail item index (exceeds number of items in the maildrop box)
NX_POP3_INSUFFICIENT_PACKET_PAYLOAD	Client packet payload too small for POP3 request
NX_POP3_SERVER_ERROR_STATUS	Server replies with error status
NX_POP3_CLIENT_INVALID_STATE	Client not initialized to receive mail messages

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

*These are error codes which are only returned if error checking is enabled. Please refer to the NetX User Guide for the Renesas Synergy™ Platform for more details on error checking services in NetX.

4.3.28.3 NetX/NetX Duo POP3 Client Module Operational Overview

The POP3 protocol requires that Clients maintain the state of the POP3 session. The POP3 Client has three distinct states defined by RFC 1939. The initial state is the authorization state in which it must identify itself to the Server. Then the Client enters the Transaction state where the Client downloads mail. When the POP3 Client chooses to end the session, it enters the Update state to disconnect from the Server.

The NetX POP3 Client requires a previously created NetX IP instance and a packet pool to send POP3 messages to the Server. The NetX POP3 Client needs the packet pool for sending out POP3 messages. It can use the same packet pool used by the IP instance or NetX can create a separate packet pool. POP3 Client messages to the Server are limited to simple POP3 commands and login authentication data. This means that if the NetX POP3 Client is using its own packet pool, it need not set the payload size much greater than 150-200 bytes, depending on the length of POP3 Client username and password. Note that the packet pool used by the IP instance, however, must have a payload large enough for receiving POP3 mail data up to the device MTU (typically 1518 bytes).

Because the NetX POP3 Client utilizes TCP services, the TCP must be enabled on the IP instance.

NetX/NetX Duo POP3 Client Module Important Operational Notes and Limitations

NetX/NetX Duo POP3 Client Module Operational Notes

The NetX Client POP3 is compliant with RFC 1939.

The auto generated Synergy code takes care of these tasks at startup/initialization. IP instance and packet pool creation, as well as enabling TCP services, is done automatically.

The POP3 Client property *Auto initialization* defaults to Disable. You must not enable this property, because the auto generated code will try to connect to the server during initialization before the network link is enabled, and fail. This in turn will abort the application initialization.

So with *Auto initialization* disabled, the application handles connecting to the Server and retrieving its mail.

The following properties are necessary for connecting to the Server:

- APOP authentication: defaults to Disable. This enables username and password to be exchanged in encrypted text and is recommended to avoid exposing this data.
- Server port number: defaults to 110, as recommended in RFC 1939, but can be any value.
- *Server_IPv4 Address: This is the server to connect to if using IPv4 connections*
- Server_IPv6 *Address: This is the server to connect to if using IPv6 connections*
- User server address type: set to IPv4 or IPv6 depending on which your application will use
- Client Name: user name to identify the POP3 client to the server
- Client Password: password which has been previously agreed to if the server requires one

However, because of the structure of the POP3 Client, setting these properties are not accessible to the application. Your application must connect to the server using the nx_pop3_client_create API and input these properties directly.

```
UINT  nx_pop3_client_create(NX_POP3_CLIENT *client_ptr,
                            UINT APOP_authentication, NX_IP *ip_ptr,
                            NX_PACKET_POOL *packet_pool_ptr,
                            ULONG server_ip_address,
                            ULONG server_port,
                            CHAR *client_name,
                            CHAR *client_password)
```

For most applications a typical call might look like:

```
Status = nx_pop3_client_create(&g_pop3_client0, NX_TRUE, &g_ip0,
                            &g_packet_pool0, /* If sharing IP packet pool */
                            IP_ADDRESS(192,168,0,2,
                            110,
                            "myusername",
                            "mypassword")
```

For NetX Duo, the ULONG server_ip_address is replaced by a pointer to an NXD_ADDRESS type:

```
NXD_ADDRESS server_address;
server_address.nxd_ip_version = NX_IP_VERSION_IPv4;  /* e.g. 4 */
/* If the local IP address is 192.168.0.2, use the IP_ADDRESS macro to convert to a
ULONG */
server_address.nxd_ip_address.v4 = IP_ADDRESS(192,168,0,);
status =  nxd_pop3_client_create(&g_pop3_client0, NX_TRUE, &g_ip0,
                            &g_packet_pool0, /* If sharing IP packet pool */
                            &server_address,
                            110,
                            "myusername",
                            "mypassword")
```

This function creates the POP3 Client instance, creates a TCP socket and binds to a local TCP port. Then it attempts to connect to the POP3 Server TCP socket. If this is successful, then it authenticates itself with the user password to the server. More details on POP3 authentication are described later in this section. If all goes well, the POP3 Client is now ready to get its mail.

The application can query the server for how many mail items are in its inbox with the nx_pop3_client*_*mail_items_get API. This will also return the total number of bytes of mail messages, although this should be taken as an estimate. The application can further query the size of each mail item, using the nx_pop3_client_mail_item_size_get API, and again this should be taken as an estimate as the actual size might vary somewhat. Note that the first mail item in the maildrop is at index 1, not zero.

To actually download the mail item, the application indicates to the server which mail item it wants to receive using the nx_pop3_client_get_mail_item service, specifying the index of the mail item. This will also return the mail item size but now the POP3 client can get the message text by calling nx_pop3_client_mail_item_get_message_data. This API will receive one packet at a time. So the application must call it one or more times until the last packet containing the message is received. The Server will indicate the last packet to the Client in the final_packet pointer as shown below:

```
    /* Find out how many items are in our mailbox.  */
    status = nx_pop3_client_mail_items_get(&demo_client, &number_mail_items,
                                           &total_size);
    mail_item = 1;
    /* Download all mail items.  */
    while (mail_item <= number_mail_items)
```

```
    {
        /* Submit a request for the next mail item. */
        status = nx_pop3_client_mail_item_get(&demo_client, mail_item,
                                              &mail_item_size);
        /* Loop to get the next mail message until it is completely downloaded. */
        do
        {
            status = nx_pop3_client_mail_item_message_get(&demo_client, &packet_ptr,



                                                  &bytes_retrieved,
&final_packet);
            /* Determine if this is the last data packet. */
            if (final_packet)
                 status = nx_pop3_client_mail_item_delete(&demo_client, mail_item);
        } while (final_packet == NX_FALSE);
        /* Get the next mail item. */
        mail_item++;
    }
```

After receiving each packet, the application should copy the packet data to a buffer and release the packet back to the packet pool (not shown above). This prevents depleting the packet pool used to receive packets. After downloading a message, it is common practice to mark the mail item for deletion. To do so, the application calls nx_pop3_client_mail_item_delete with the index of the mail item to delete.

When done downloading mail items, the Client quits the session by calling nx_pop3_client_quit. This terminates the TCP session. If no longer using POP3, the application can delete the TCP socket by calling nx_pop3_client_delete. If no other tasks are using the POP3 Client packet pool, the application can delete the packet pool using the nx_packet_pool_delete service.

POP3 Authentication

After the TCP connection is established, a POP3 client needs to identify itself to the POP3 server through authentication to access its mailbox. Authentication is accomplished by sending the server a username and password. The username is typically a fully qualified domain name (contains a local part and a domain name, separated by an '@' character.

To enable APOP Authentication, set the *APOP Authentication* input in the nx_pop3_client_create/nxd_pop3_client_create API to NX_TRUE (or nxd_pop3_client_create API in NetX Duo). APOP Authentication creates an MD5 digest of user name/password supplied to the nx_pop3_client_create/nxd_pop3_client_create call and avoids the security risk of transmitting

username and password in clear text. If APOP authentication fails, the NetX POP3 Client will attempt to login using the username and password without encryption.

NetX/NetX Duo POP3 Client Module Limitations

- The NetX POP3 Client does not support the AUTH command but does support APOP authentication using DIGEST MD5.
- NetX POP3 Client does not implement all POP3 commands (for example, the TOP or UIDL commands).
- See the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.28.4 Including the NetX/NetX Duo POP3 Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo POP3 Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo POP3 Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo POP3 Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_pop3_client0 NetX POP3 Client	Threads	New Stack> X-Ware> NetX> Protocols> NetX POP3 Client
g_pop3_client0 NetX POP3 Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo POP3 Client

When the NetX and/or NetX Duo POP3 Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

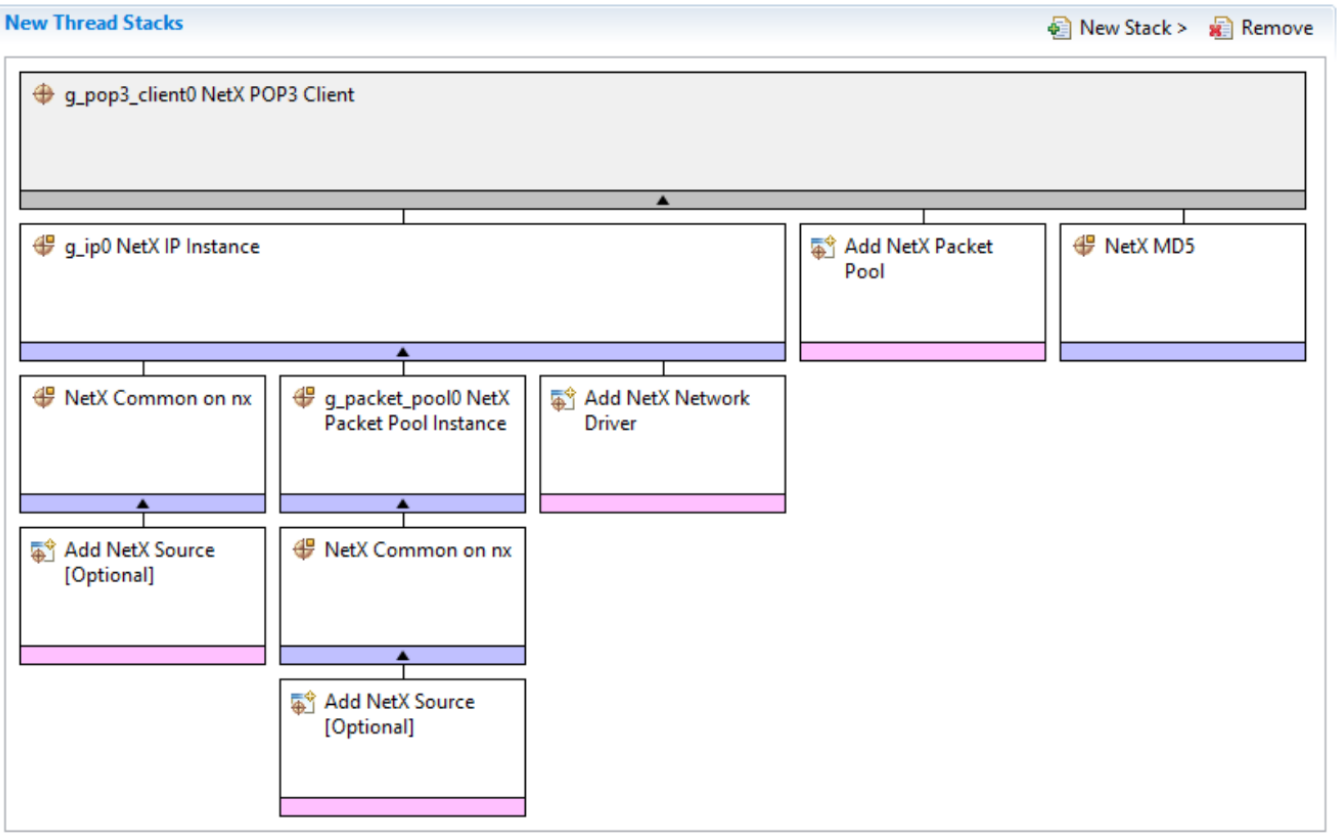


Figure 487: NetX/NetX Duo POP3 Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.28.5 Configuring the NetX/NetX Duo POP3 Client Module

The NetX/NetX Duo POP3 Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo POP3 Client Module

ISDE Property	Value	Description
Maximum buffer size to store messages (bytes)	2000	Maximum buffer size to store messages selection
Packet time out (seconds)	1	Packet time out selection
Connection time out (seconds)	30	Connection time out selection
Disconnect time out (seconds)	2	Disconnect time out selection
TCP socket send wait (seconds)	2	TCP socket send wait selection
Server reply timeout (seconds)	10	Server reply timeout selection
TCP window size (bytes)	1460	TCP window size selection
Maximum user name length (bytes)	40	Maximum user name length selection
Maximum password length (bytes)	20	Maximum password length selection
Name	g_pop3_client0	Module name
APOP Authentication	Enable, Disable Default: Disable	Apop authentication selection
**Use server address type	IPv4, IPv6 Default: IPv6	Use server address type selection
Server IPv4 Address (use commas for separation)	192, 168, 0, 2	Server IPv4 Address selection
**Server IPv6 Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	Server IPv6 Address selection
Server Port number	110	Server Port number selection
Client Name	username@domain.com	Client name selection
Client Password	password	Client password selection
Auto initialization	Enable, Disable Default: Disable	Auto initialization selection
Name of generalized initialization function	pop3_client_init0	Name of generalized initialization function selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC or IP Addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo POP3 Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection

UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection

Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5 Instance

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo POP3 Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo POP3 Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1

Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.28.6 Using the NetX/NetX Duo POP3 Client Module in an Application

The steps in using the NetX and NetX Duo BSD Support module in a typical application are:

Step 1. Wait for the network link to be enabled with the nx_ip_status_check API.

Step 2. Create the POP3 Client using nx_pop3_client_create API (nxd_pop3_client_create in NetX Duo POP3 Client).

Step 3. Send a request to the POP3 Server for the number of mail items in mailbox with the nx_pop3_client_mail_items_get API.

Step 4. Assuming there are one or more items in the POP3 Client maildrop, send a request to the POP3 Server one or all of them by calling the nx_pop3_client_mail_item_get API.

Step 5. After each call to nx_pop3_client_mail_item_get, download the actual mail message data using the nx_pop3_client_mail_message_get API.

Step 6. Copy the packet data into a separate buffer and release the receive packet(s) using the nx_packet_release API [Strongly recommended to avoid packet pool depletion]

Step 7. Check if this is the last packet of the message: the final_packet pointer input of nx_pop3_client_mail_message_get will be TRUE if it is the last packet.

If FALSE, call nx_pop3_client_mail_message_get again. If TRUE, go on to step 8.

Step 8. Mark the current mail item for deletion using the nx_pop3_client_mail_item_delete API. This will send a DELE message to the server for it to delete the mail item at some later time.

Step 9. Check if there are more mail items to download. If yes, get the next mail item by calling

nx_pop3_client_mail_item_get. If no, go on to step 10.

Step 10. Send a QUIT message to the Server with the nx_pop3_client_delete API

Step 11. Delete the POP3 instance by calling nx_pop3_client_delete API.

The following figure illustrates common steps in a typical operational flow diagram:

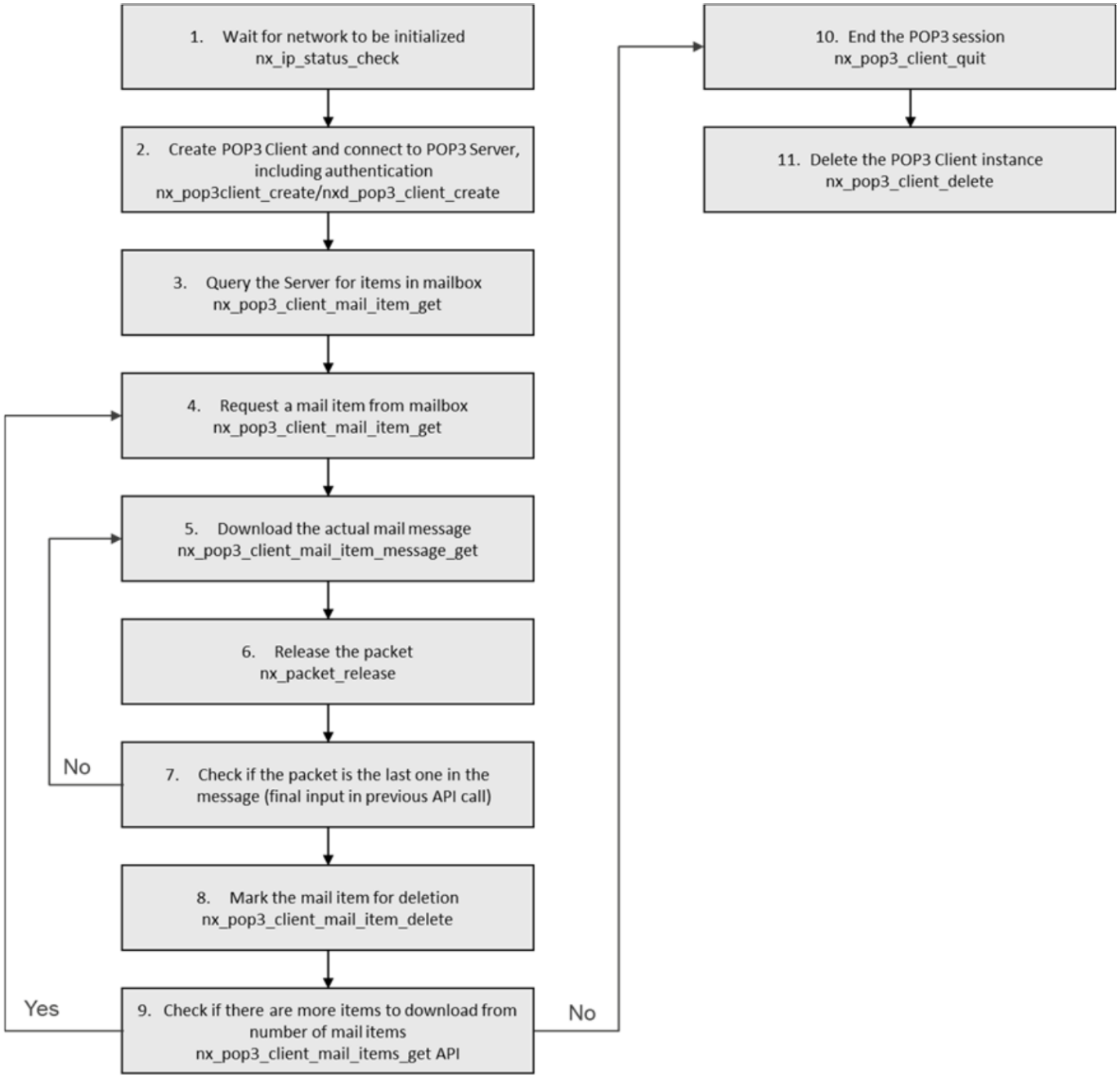


Figure 488: Flow Diagram of a Typical NetX/NetX Duo POP3 Client Module Application

4.3.29 NetX/NetX Duo Telnet Client

4.3.29.1 NetX and NetX Duo Telnet Client Introduction

The Telnet Protocol (Telnet) protocol is designed for transferring commands and responses between two nodes on the Internet. Telnet is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its transfer function. The NetX™ Telnet Client provides a high-level API for applications wishing to implement a Telnet Client.

Note

Except for internal processing, the NetX Duo™ Telnet Client is nearly identical in the application, set up and running a Telnet session as the NetX™ Telnet Client (except where noted).

Unsupported Features

Multi-thread support has not been tested in this version of SSP.

NetX and NetX Duo Telnet Client Module Features

- The NetX™ Telnet Client Module is compliant with RFC854 and related RFCs
- Provides high-level APIs to:
    - Create and delete a Telnet Client instance
    - Connect and disconnect a Telnet Client instance
    - Send to and receive packets from a Telnet Server
    - Support multi-thread operation

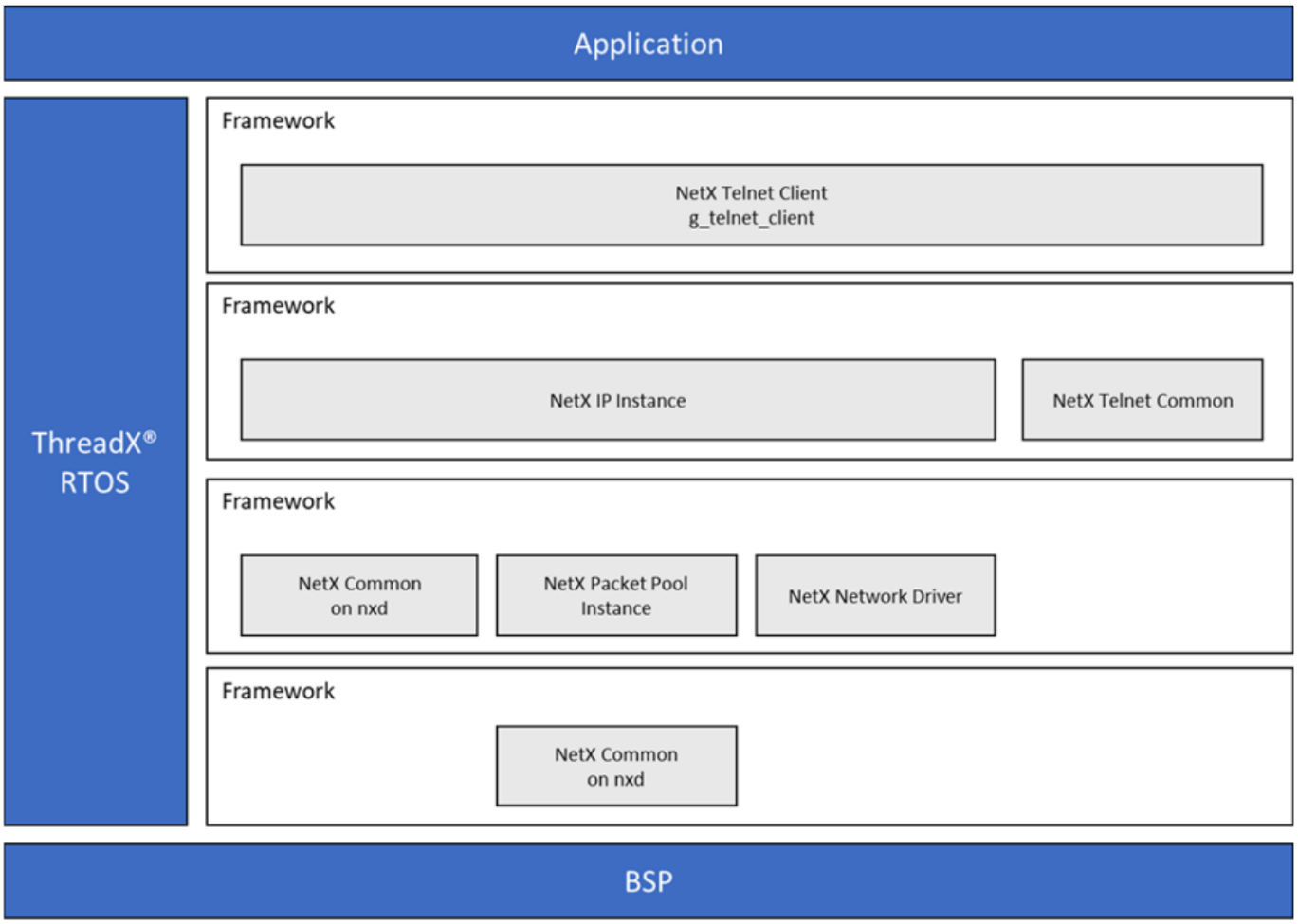


Figure 489: NetX and NetX Duo Telnet Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX and NetX Duo Telnet Client Module in an Application for additional details.

4.3.29.2 NetX and NetX Duo Telnet Client Module APIs Overview

The NetX™ Telnet Client Module defines APIs for creating, deleting, connecting, disconnecting, receiving and sending telnet communications. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX and NetX Duo Telnet Client Module API Summary

Function Name	Example API Call and Description
nx_telnet_client_connect	nx_telnet_client_connect(&g_telnet_client0, server_ip_address, server_port, NX_WAIT_FOREVER); Connect a Telnet Server. Supports only IPv4.

**nxd_telnet_client_connect	nxd_telnet_client_connect(&g_telnet_client0, &server_ip_address_v6, server_port, NX_WAIT_FOREVER); Connect to a Telnet Server. Supports IPv4 and IPv6.
nx_telnet_client_create	nx_telnet_client_create(&g_telnet_client0, "Telnet Client", &g_ip0, 1024); Create a Telnet Client.
nx_telnet_client_delete	nx_telnet_client_delete(&g_telnet_client0); Delete a Telnet Client.
nx_telnet_client_disconnect	nx_telnet_client_disconnect(&g_telnet_client0, NX_WAIT_FOREVER); Disconnect from the Telnet Server (IPv4 or IPv6).
nx_telnet_client_packet_receive	nx_telnet_client_packet_receive(&g_telnet_client0, &packet_ptr, NX_WAIT_FOREVER); Receive packet from the Telnet Server.
nx_telnet_client_packet_send	nx_telnet_client_packet_send(&g_telnet_client0, packet_ptr, NX_WAIT_FOREVER); Send packet to the Telnet Server (IPv4 or IPv6).

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo Telnet Client. For definitions of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_TELNET_ERROR	Error while creating TCP socket as part of creating the Telnet Client instance
NX_TELNET_NOT_CONNECTED	Client not disconnected
NX_TELNET_NOT_DISCONNECTED	Client socket is not in closed state (cannot make a TCP connection; cannot delete the Telnet Client if the socket is still connected).
NX_TELNET_INVALID_PARAMETER*	Invalid non-pointer input to Telnet Client create
NX_PTR_ERROR*	Invalid pointer input
NX_IP_ADDRESS_ERROR*	Invalid IP address to connect to Telnet Server
NX_CALLER_ERROR*	Invalid caller of this service

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

*These error codes are only returned if error checking is enabled. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for more details on error checking services in NetX Duo.

4.3.29.3 NetX and NetX Duo Telnet Client Module Operational Overview

In the NetX™ Telnet Client Module, the IP thread task for NetX™ is created and runs, the Telnet Client is created and the TCP socket for connecting to the Telnet Server is ready for use automatically. The Telnet Client connects to the server by calling the nx_telnet_client_connect API. (This API is available in NetX Duo™ Telnet and is limited to IPv4 TCP connections. In NetX Duo™, the application can also use the nxd_telnet_client_connect which supports both IPv4 and IPv6.) If that succeeds, then the Telnet Client should wait to receive a Server 'banner' announcing itself using the nx_telnet_client_packet_receive API. Thereafter the Telnet Client can create packets with single characters of data using the nx_packet_allocate and nx_packet_data_append API. These packets are sent to the Telnet Server by calling the nx_telnet_client_packet_send API.

For a more complete description of the NetX™ Telnet Client and NetX Duo™ Telnet Client operations, refer to the associated User Guides found in the Synergy Gallery, under the SSP Documentation tab as "Azure RTOS and NetX™ Component Documentation for Renesas Synergy". Open the zip file and navigate to the related User Guides for detailed descriptions. This document provides only an introduction to component operations. Module selection, configuration, and example uses are described in detail in the following sections.

NetX and NetX Duo Telnet Client Module Important Operational Notes and Limitations

NetX and NetX Duo Telnet Client Module Operational Notes

- The NetX Duo™ Telnet Client Module services can be called from multiple threads simultaneously. However, read or write requests for a particular Telnet Client instance should be done in sequence from the same thread.

NetX and NetX Duo Telnet Client Module Limitations

- The Telnet Client does not support Telnet negotiation or send IAC and command code sequences.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.29.4 Including the NetX and NetX Duo Telnet Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo Telnet Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX and NetX Duo Telnet Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX and NetX Duo Telnet Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_telnet_client0 NetX™ Telnet Client	Threads	New Stack> X-Ware> NetX™ > Protocols> NetX™ Telnet Client
g_telnet_client0 NetX Duo™ Telnet Client	Threads	New Stack> X-Ware> NetX Duo™ > Protocols> NetX Duo™ Telnet Client

When the NetX and/or NetX Duo Telnet Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

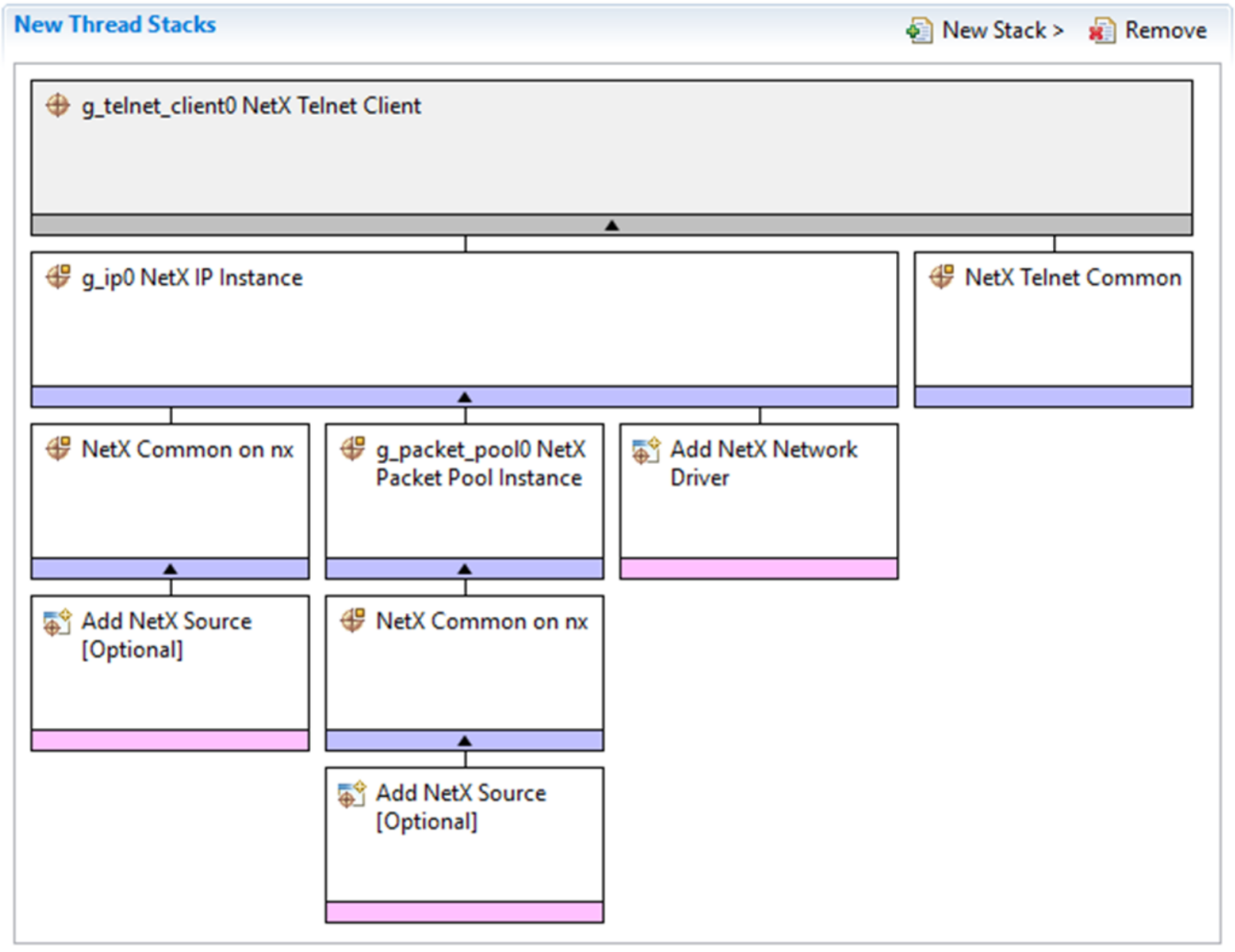


Figure 490: NetX and NetX Duo Telnet Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not

all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.29.5 Configuring the NetX and NetX Duo Telnet Client Module

The NetX and NetX Duo Telnet Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX and NetX Duo Telnet Client Module

ISDE Property	Value	Description
Name	g_telnet_client0	Module name
TCP Socket Window Size in Bytes	1024	TCP socket window size selection
Name of generated initialization function	telnet_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different IP addresses and subnet masks. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX and NetX Duo Telnet Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX and NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection

IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX and NetX Duo Telnet Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection
Server TCP port number	23	Server TCP port number selection
Time to live	128	Time to live selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have

different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX and NetX Duo Telnet Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX and NetX Duo Telnet Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description

Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.29.6 Using the NetX and NetX Duo Telnet Client Module in an Application

The NetX™ Telnet Client Module does not need the usual initialization by an application. A configurator generates initialization process. The user application only needs Telnet communication processing.

The steps in using the NetX and NetX Duo Telnet Client module in a typical application are:

1. Use the nx_ip_status_check API to check that the IP instance is initialized and the application can start using NetX™ services.

2. Connect to the Telnet Server via the nx_telnet_client_connect API. (Note: For NetX Duo™ the preferred API is nxd_telnet_client_connect).

3. Receive the Telnet Server welcome banner using nx_telnet_client_packet_receive API [Optional]

4. Create a packet to send with the nx_packet_allocate and nx_packet_data_append APIs.

5. Send the packet using the nx_telnet_client_packet_send API

6. Receive the Server's response packet using the nx_telnet_client_receive API.

7. Disconnect communication using the nx_telnet_client_disconnect API

8. Delete the instance using the nx_telnet_client_delete API when done sending and receiving.

The following figure illustrates common steps in a typical operational flow diagram:

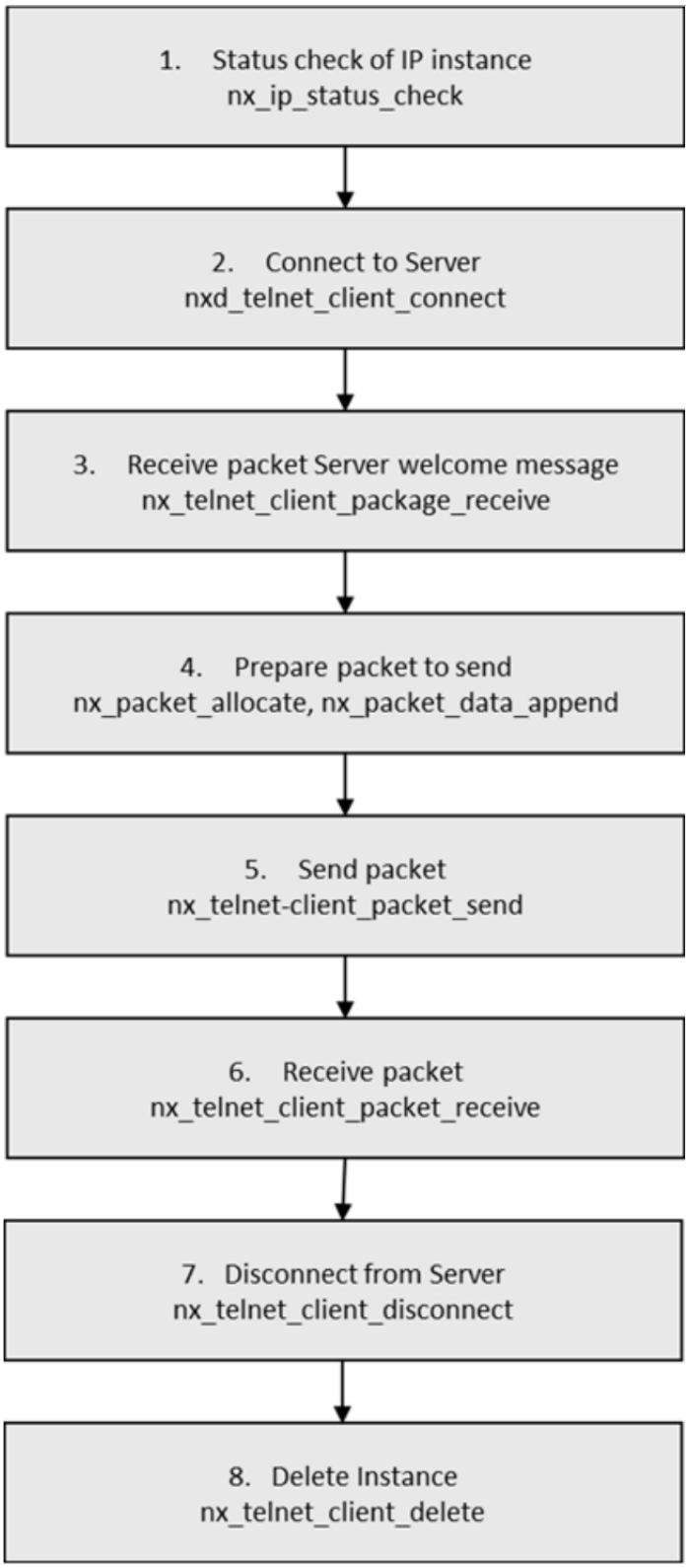


Figure 491: Flow Diagram of a Typical NetX and NetX Duo Telnet Client Module Application

4.3.30 NetX/NetX Duo Telnet Server

4.3.30.1 NetX and NetX Duo Telnet Server Introduction

The Telnet Protocol (Telnet) is a protocol designed for transferring commands and responses between two nodes on the internet. Telnet is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its transfer function.

Note

Except for internal processing, the NetX Duo™ Telnet Server is nearly identical in the application, setup and running of a Telnet session as the NetX™ Telnet Server (except where noted).

Unsupported Features

Telnet Option Negotiation has not been tested for NetX Duo Telnet Server in this version of SSP

NetX and NetX Duo Telnet Server Module Features

- The NetX Telnet Server module is compliant with RFC854 and related RFCs.
- Provides high-level APIs to:
    - Support multi-thread operation
    - Support callbacks for common functions
    - Authentication
    - New Connection
    - Receive Data
    - End Connection
- Supports some option negotiation
    - Echo
    - Suppress Go Ahead

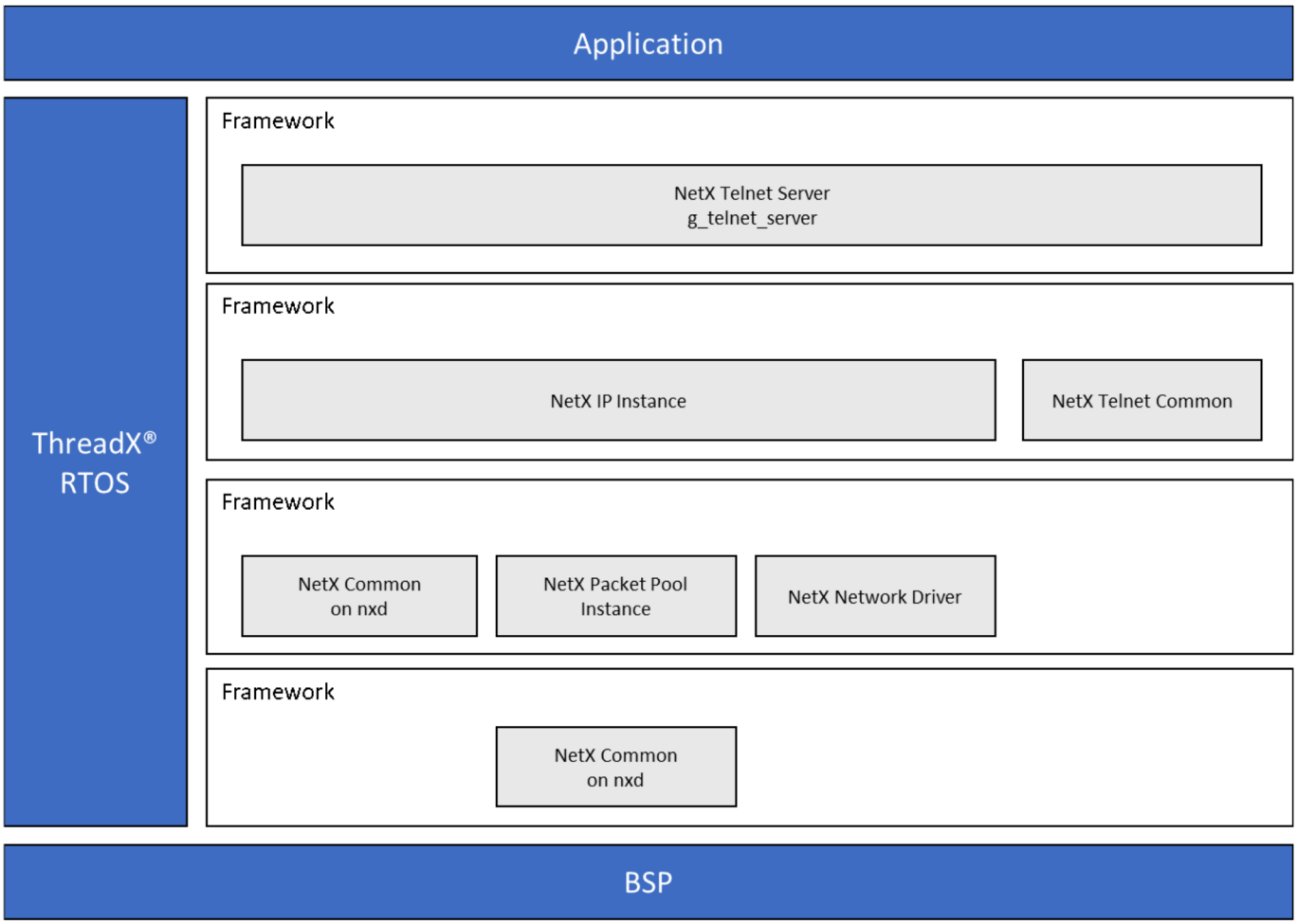


Figure 492: NetX and NetX Duo Telnet Server Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX and NetX Duo Telnet Server Module in an Application for additional details.

4.3.30.2 NetX and NetX Duo Telnet Server Module APIs Overview

The NetX Telnet Server module defines APIs for creating, deleting, sending packets, starting and stopping. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX and NetX Duo Telnet Server Module API Summary

Function Name	Example API Call and Description
nx_telnet_server_create	nx_telnet_server_create(&my_server, "Telnet Server", &ip_0,pointer, 2048, telnet_new_connection, telnet_receive_data, telnet_connection_end); Create a Telnet Server.

nx_telnet_server_delete	nx_telnet_server_delete(&my_server); Delete a Telnet Server.
nx_telnet_server_disconnect	nx_telnet_server_disconnect(&my_server, 2); Disconnect the Telnet Client specified by the client list index (2nd input).
nx_telnet_server_get_open_connection_count	nx_telnet_server_get_open_connection_count(&my_server, &conn_count); Retrieve the number of open connections.
nx_telnet_server_packet_send	nx_telnet_server_packet_send(&my_server, 2, my_packet, 100); Send packet to Telnet Client specified by client list index (second input).
nx_telnet_packet_pool_set	nx_telnet_server_packet_pool_set(&my_server, &telnet_server_packet_pool); Set packet pool as Telnet Server packet pool.
nx_telnet_server_start	nx_telnet_server_start(&my_server); Start theTelnet Server.
nx_telnet_server_stop	nx_telnet_server_stop(&my_server); Stop the Telnet Server.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful telnet function
NX_PTR_ERROR*	Invalid Server, IP, stack, or application callback pointers
NX_CALLER_ERROR*	Invalid caller of this service
NX_OPTION_ERROR*	Invalid logical connection
NX_IP_ADDRESS_ERROR*	Invalid IP address
NX_TELNET_FAILED	Server packet send failed
NX_TELNET_NO_PACKET_POOL	Cannot start Telnet Server, no packet pool available
NX_TELNET_NOT_CONNECTED	Cannot disconnect Telnet Server because it is not connected
NX_TELNET_NOT_DISCONNECTED	Cannot connect or delete Telnet Server because it is not disconnected

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

*These error codes are only returned if error checking is enabled. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for more details on error checking services in NetX Duo.

4.3.30.3 NetX and NetX Duo Telnet Server Module Operational Overview

The NetX Telnet package requires that a NetX IP instance has already been created. In addition, TCP must be enabled on that same IP instance. The Telnet Client portion of the NetX Telnet package has no further requirements. It also requires complete access to TCP well-known port 23 for handling all Telnet Client requests. The Telnet Server keeps a list of client connections and uses an index into this list to specify certain clients when needed. The size of this list is set in the Maximum clients to server simultaneously property.

The Telnet Server can be enabled for limited Option negotiation with the Telnet Client. To enable this feature, set the Option negotiation to enable. If this feature is enabled, the Telnet Server needs a packet pool. The application can set the packet payload and number of packets; packet_size_inthe pool and Totalpacket_pool_sizeProperties, respectively and let the Telnet Server create the packet pool. When the Telnet Server is deleted, the packet pool is deleted with it.

Alternatively, it can create the packet pool directly and be set as the Telnet Server packet pool by 1) enabling Use application packet pool or by 2) creating the packet pool by calling the NetXnx_packet_pool_create API and 3) setting the packet pool in the Telnet Server using the nx_telnet_server_packet_pool_set API. When the Telnet Server is deleted, the application must delete the packet pool directly (nx_packet_pool_delete API).

Telnet New Connection Callback

The NetX Telnet Server calls the new connection callback function when a new Telnet Client request is received. The callback function is set in the NetX Telnet Server Name of Client Connect Callback Function property. Actions of the new connection callback include sending a banner or prompt to the client. It could also include a prompt for login information if authentication is required. The second argument of the new connection callback specifies the client is connecting.

Telnet Receive Data Callback

The NetX Telnet Server Module calls the data received callback function when a new Telnet Client data is received. The second input of the callback is an index into the Telnet Server's list of clients so the Telnet Server knows which client wants to disconnect. The callback function is set in the Name of Receive Data Callback Function property. Typical actions of the receive data callback include echoing the data back and/or parsing the data and providing data because of interpreting a command from the client.

Note

This callback routine must release the received packet.

Telnet End Connection Callback

The NetX Telnet Server calls the end connection callback function when it receives a Telnet Client disconnect request. The second input of the callback is an index into the Telnet Server's list of clients so the Telnet Server knows which Client wants to disconnect. The callback function is set in the Name of Client Disconnect Callback Function property. Typical actions of the end connection callback include cleaning up resources used for the Telnet Client session.

Telnet Option Negotiation

Upon making a connection with the Telnet Client, the Telnet Server will send out this set of Telnet

options to the client if it has not received option requests from the client:

will echo

don't echo

will sga

When it receives Telnet data from the client, the Telnet Server checks if the first byte is the IAC (Interpret as Command) code; if so, it will process all the options in the client packet. Options not in the list above are not supported and will be ignored.

NetX and NetX Duo Telnet Server Module Important Operational Notes and Limitations

NetX and NetX Duo Telnet Server Module Operational Notes

- For the connect callback, the application can use any packet pool it has created or the IP default packet pool. If nx_telnet_server_packet_send fails, the callback must release that packet.
- The number of active client connections can be obtained at any time by calling the nx_telnet_server_get_open_connection_count API.
- The Telnet Server thread task periodically checks the time remaining on each client connection inactivity timeout. If the timeout has expired, the client connection is dropped. To set the length of the inactivity timeout, set the value of the Client inactivity timeout property of the Telnet Server to the desired value. The interval that the Telnet Server thread task checks the inactivity timeout is the Timeout check period property; this must be less than the client inactivity timeout.
- The Telnet Server can be stopped using the nx_telnet_server_stop API and restarted using the nx_telnet_server_start API. When the Telnet Server is stopped, all client connections are dropped and the server stops listening on the Telnet port.
- Deleting the Telnet Server is like stopping the Telnet Server, but additionally releases all resources used for the Telnet Server; timer, thread, TCP socket and if created by the Telnet Server, the Telnet Packet pool.
- The interpretation and response to Telnet Client commands, indicated by a byte with the value of 255, is the responsibility of the application.

NetX and NetX Duo Telnet Server Module Limitations

- The NetX Telnet Server supports only a limited set of Telnet Option commands.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.30.4 Including the NetX and NetX Duo Telnet Server Module in an Application

This section describes how to include either or both the NetX and NetX Duo Telnet Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX and NetX Duo Telnet Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX and NetX Duo Telnet Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_telnet_server0 NetX Telnet Server	Threads	New Stack> X-Ware> NetX> Protocols> NetX Telnet Server
g_telnet_server0 NetX Duo Telnet Server	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo Telnet Server

When the NetX and/or NetX Duo Telnet Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

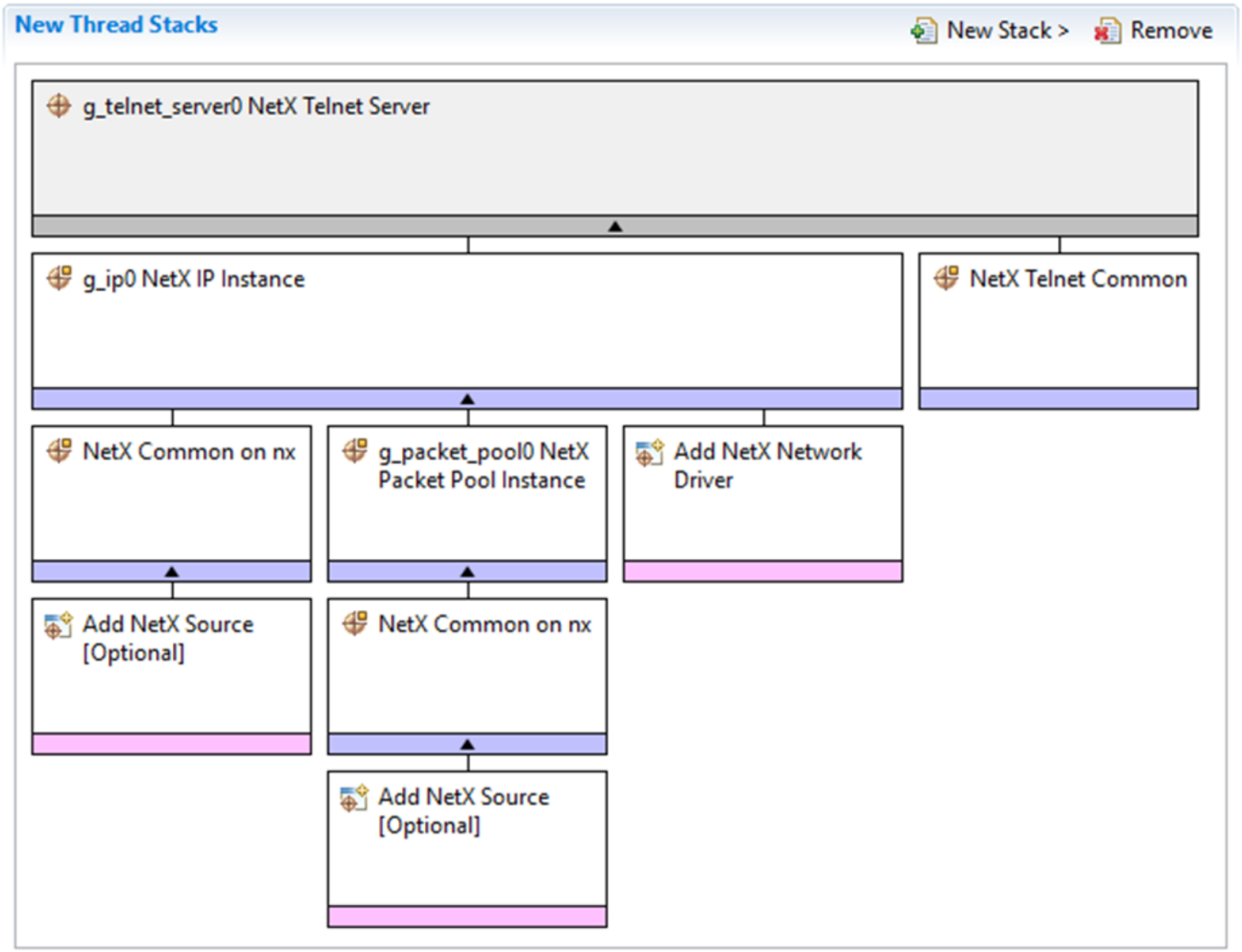


Figure 493: NetX and NetX Duo Telnet Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.30.5 Configuring the NetX and NetX Duo Telnet Server Module

The NetX and NetX Duo Telnet Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX and NetX Duo Telnet Server Module

ISDE Property	Value	Description
Internal thread priority	16	Internal thread priority selection
Maximum clients to serve simultaneously	4	Maximum clients to serve simultaneously selection
Socket window size (bytes)	2048	Socket window size (bytes) selection
Server time out (seconds)	10	Server time out (seconds) selection
Client inactivity timeout (seconds)	600	Client inactivity timeout (seconds) selection
Timeout check period (seconds)	60	Timeout check period (seconds) selection
Option negotiation	Enable, Disable Default: Enable	Option negotiation selection
Use application packet pool	Enable, Disable Default: Disable	Use application packet pool selection

Packet size in the pool (bytes)	300	Packet size in the pool (bytes) selection
Total packet pool size (bytes)	2048	Total packet pool size (bytes) selection
Name	g_telnet_server0	Name selection
Internal thread stack size (bytes)	2048	Internal thread stack size (bytes) selection
Name of Client Connect Callback Function	telnet_client_connect	Name of Client Connect Callback Function selection
Name of Receive Data Callback Function	telnet_receive_data	Name of Receive Data Callback Function selection
Name of Client Disconnect Callback Function	telnet_client_disconnect	Name of Client Disconnect Callback Function selection
Name of generated initialization function	telnet_server_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different IP addresses and subnet masks. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX and NetX Duo Telnet Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX and NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection

Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX and NetX Duo Telnet Common Instance

ISDE Property	Value	Description
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection
Server TCP port number	23	Server TCP port number selection
Time to live	128	Time to live selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX and NetX Duo Telnet Server Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX and NetX Duo Telnet Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin

MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.30.6 Using the NetX and NetX Duo Telnet Server Module in an Application

The steps in using the NetX and NetX Duo Telnet Server module in a typical application are:

1. Use the nx_ip_status_check API to check that the IP instance can communicate.

2. Configure the Telnet Server to do option negotiation [optional].

3. If options negotiation is enabled, configure the Telnet Server to create its own packet pool (default) or set the Telnet Server pool from the application.

4. Start the Telnet Server using the nx_telnet_server_start API.

5. Process client requests with the callbacks specified at build time.

6. Delete the Telnet Server when done.

7. If the Telnet Server packet pool was created by the application, it can be deleted if not in use by other threads.

The NetX Telnet Server module application needs to execute response processing in the registered callback function. Typically, in callback processing, the content of the received packet is confirmed and data content is transmitted with the nx_telnet_server_packet_send API. When disconnection of communication is required from the server side, call the nx_telnet_server_disconnect API.

The following figure illustrates common steps in a typical operational flow diagram:

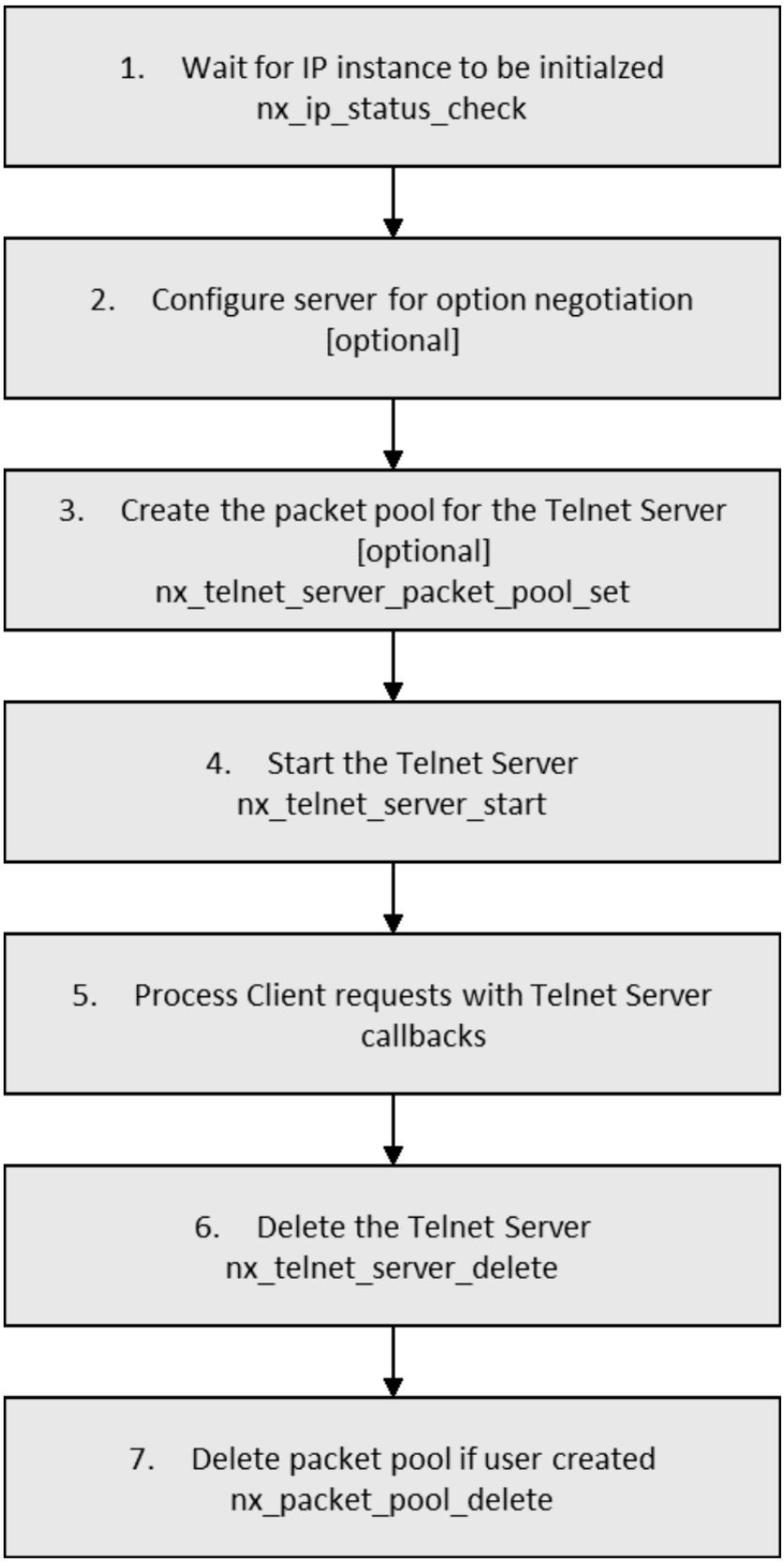


Figure 494: Flow Diagram of a Typical NetX and NetX Duo Telnet Server Module Application

4.3.31 NetX/NetX Duo TFTP Client

4.3.31.1 NetX/NetX Duo TFTP Client Introduction

The Trivial File Transfer Protocol (TFTP) is a lightweight protocol designed for file transfers over UDP. Unlike more robust TCP based protocols, TFTP does not perform extensive error checking and can have limited performance because it is a stop-and-wait protocol. After a TFTP data packet is sent, the sender waits for an ACK to be returned by the recipient. Although this is simple, it does limit the overall TFTP throughput. The TFTP Server utilizes the well-known UDP port 69 to listen for client requests. TFTP Clients may use any available UDP port. Data packets are 512 bytes, until the last packet. A packet containing fewer than 512 bytes signals the end of file.

Note

Except where noted, the NetX Duo TFT Client is identical in application set up and running of a TFTP session as the NetX TFTP Client. For setting up the IP instance for IPv6 in NetX Duo, please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform.

Unsupported Features

Multi-thread support has not been tested in this version of SSP.

NetX/NetX Duo TFTP Client Module Features

- The NetX TFTP Client Module is compliant with RFC 1350 and related RFCs.
- High level APIs for:
  - Creating and deleting a TFTP client
  - Reading and writing files to TFTP recipient (server)
  - Set network interface for TFTP Client to run on

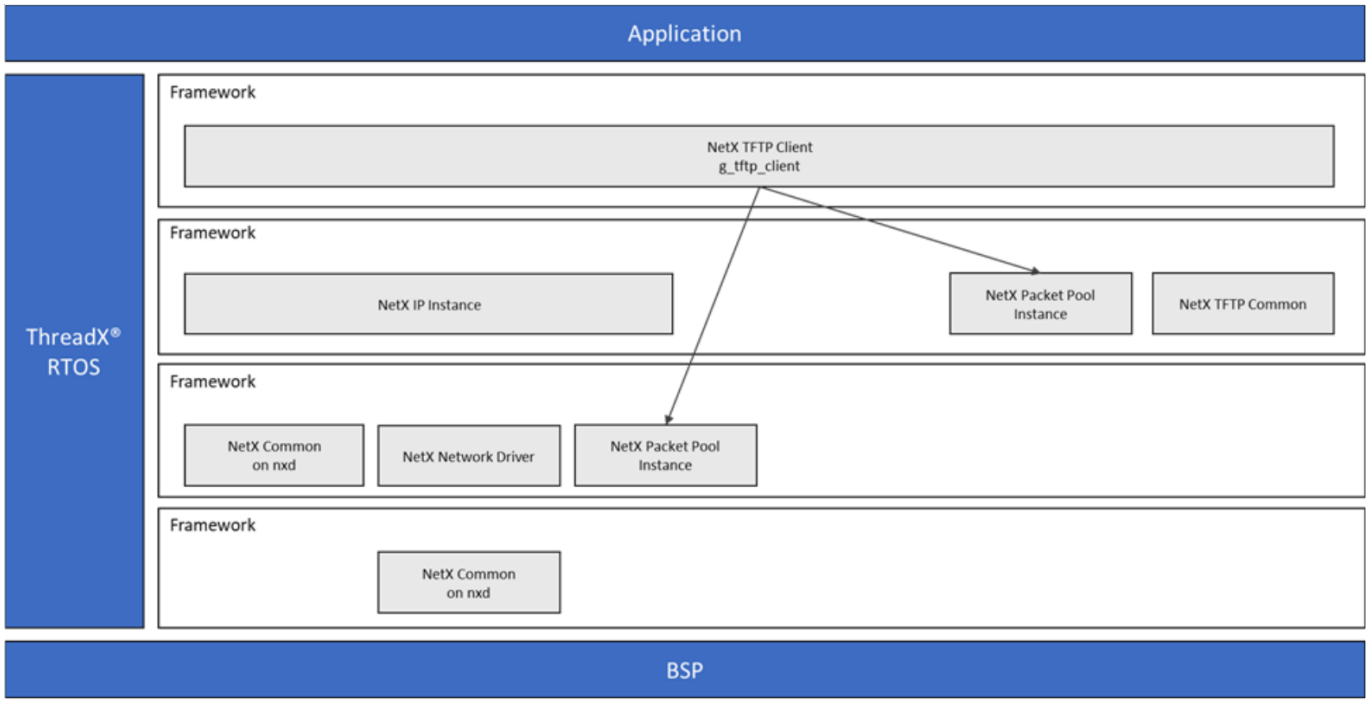


Figure 495: NetX/NetX Duo TFTP Client Module Block Diagram

Note

In the figure above, the NetX (or NetX Duo) Network Driver module has multiple implementation options available. See the description just after the module stack figure in Including the NetX/NetX Duo TFTP Client Module in an Application for additional details.

4.3.31.2 NetX/NetX Duo TFTP Client Module APIs Overview

The NetX/NetX Duo TFTP Client module defines APIs for creating and starting the TFTP Client. Internally, the TFTP Client handles all communication with the TFTP Server to transfer files. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX/NetX Duo TFTP Client Module API Summary

Function Name	Example API Call and Description
nx_tftp_client_file_open	nx_tftp_client_file_open(&g_tftp_client0, "test.txt", server_ip_address, NX_TFTP_OPEN_FOR_READ, NX_WAIT_FOREVER); Open TFTP client file (IPv4 only).
nxd_tftp_client_file_open**	nxd_tftp_client_file_open(&g_tftp_client0, "test.txt", &server_ip_address_v6, NX_TFTP_OPEN_FOR_READ, NX_WAIT_FOREVER, NX_IP_VERSION_V6); Open TFTP client file. Can be IPv4 type or IPv6
nx_tftp_client_create	nx_tftp_client_create(&g_tftp_client0, "TFTP Client", &g_ip0, &g_packet_pool0); Create a TFTP client instance
nxd_tftp_client_create**	nx_tftp_client_create(&g_tftp_client0, "g_tftp_client0 TFTP Client", &g_ip0, &g_packet_pool0, NX_IP_VERSION_V6); Create a TFTP client instance.
nx_tftp_client_delete	nx_tftp_client_delete(&g_tftp_client0); Delete a TFTP client instance.
nxd_tftp_client_delete**	nxd_tftp_client_delete(&g_tftp_client0); Delete a TFTP client instance.
nx_tftp_client_error_info_get	nx_tftp_client_error_info_get(&g_tftp_client0, &error_code, &error_string); Get client error information.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo TFTP Client. For definitions of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description

NX_SUCCESS	Successful TFTP create
NX_PTR_ERROR*	Invalid IP, pool, or TFTP pointer
NX_CALLER_ERROR *	Invalid caller of this service
NX_TFTP_NOT_CLOSED	Cannot open file; Client already has file open
NX_TFTP_CODE_ERROR	Error status received from TFTP server
NX_INVALID_TFTP_SERVER_ADDRESS	Invalid server address received
NX_TFTP_FAILED	Unknown code received from Server
NX_TFTP_NO_ACK_RECEIVED	No ACK received from server on read or write
NX_TFTP_INVALID_BLOCK_NUMBER	Block of data in TFTP server ACK does not match TFTP Client after read or write request.
NX_TFTP_NOT_OPEN	TFTP client not in the open state for file read or write
NX_IP_ADDRESS_ERROR*	Invalid Server IP address
NX_OPTION_ERROR*	Invalid open type
NX_INVALID_TFTP_SERVER_ADDRESS	Invalid server address received
NX_TFTP_END_OF_FILE	End of file detected (not an error) during file download (read) transfer
NX_TFTP_TIMEOUT	Timeout waiting for Server ACK of write request
NX_TFTP_INVALID_INTERFACE*	Invalid interface input

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

*These error codes are only returned if error checking is enabled. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for more details on error checking services in NetX Duo.

4.3.31.3 NetX/NetX Duo TFTP Client Module Operational Overview

In the NetX/NetX Duo TFTP Client module, the client is automatically created and the UDP socket is created for sending and receiving TFTP packets. In a TFTP Client application, the TFTP Client first 'opens' a file. If the Server returns an acknowledgment, the Client can then request a file transfer for either reading (receiving from) or writing (sending to) the TFTP Server. After each file transfer the TFTP Client closes the file.

A TFTP Server listens for Client requests on the well-known port 69. The TFTP Client socket can bind to any port. When transferring a file in TFTP, the amount of data that added to a packet can only be 512 bytes, unless there is less than 512 bytes left to send. A packet containing fewer than 512 bytes signals the end of file. The general sequence of events is as follows:

TFTP Read File Requests:

1. Client issues an "Open for Read" request with the file name and waits for the Server response.
2. Server sends the first 512 bytes of the file as 'block number 1'

3. Client receives the data, sends an ACK specifying block number is '1' , and waits for the next packet.
4. The Server sends the rest of the data incrementing the block number each time, and the Client ACKs sends an ACK with the corresponding block number each time.
5. The sequence ends when the last packet containing fewer than 512 bytes is received. nx_tftp_client_file_read returns NX_TFTP_END_OF_FILE when the last packet of a file transfer is received.

TFTP Write Requests:

1. Client issues an "Open for Write" request with the file name and waits for an ACK with a block number of 0 from the Server.
2. The Server sends an ACK with a block number of zero.
3. Client sends the first 512 bytes of the file to the Server and waits for an ACK.
4. Server sends ACK after the bytes are written.
5. The sequence ends when the Client completes sends a packet containing fewer than 512 bytes.

NetX/NetX Duo TFTP Client Module Important Operational Notes and Limitations

NetX/NetX Duo TFTP Client Module Operational Notes

- If nx_tftp_client_file_read returns NX_SUCCESS, the caller must release the packet after it is done with it.
- If nx_tftp_client_file_write returns NX_TFTP_NOT_OPEN, the caller must release the packet. In all other cases NetX or NetX TFTP Client Module will release the packet.
- For file data, any data can be used. The NetX TFTP Client Module does not make any changes to send or receive data.
- The NetX TFTP Client Module is compliant with RFC 1350 and related RFCs.

NetX/NetX Duo TFTP Client Module Limitations

- The line feed code of the text file cannot be changed; this must be processed by the user application.
- The NetX TFTP Client Module services can be called from multiple threads simultaneously. However, read or write requests for a particular client instance should be done in sequence from the same thread.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.31.4 Including the NetX/NetX Duo TFTP Client Module in an Application

This section describes how to include either or both the NetX and NetX Duo TFTP Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo TFTP Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX/NetX Duo TFTP Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_tftp0 NetXTFTP Client	Threads	New Stack> X-Ware> NetX> Protocols> NetXTFTP Client
g_tftp0 NetX Duo TFTP Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo TFTP Client

When the NetX and/or NetX Duo TFTP Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

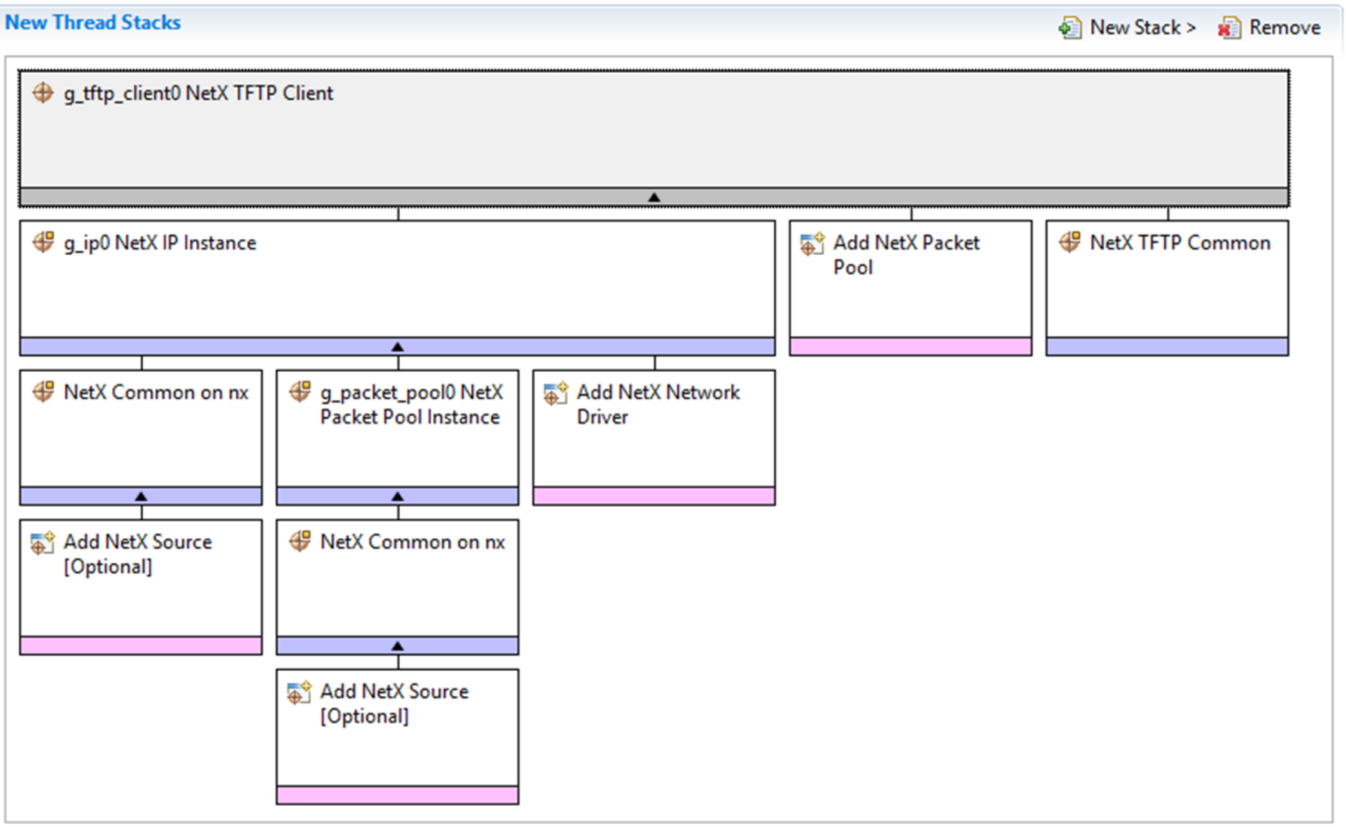


Figure 496: NetX/NetX Duo TFTP Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx

- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.31.5 Configuring the NetX/NetX Duo TFTP Client Module

The NetX/NetX Duo TFTP Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX/NetX Duo TFTP Client Module

ISDE Property	Value	Description
Source port to use	NX_ANY_PORT	Source port to use selection
Name	g_tftp_client0	Module name
Name of generated initialization function	tftp_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different pins for the Ethernet peripheral. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX/NetX Duo TFTP Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX/NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection

Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX/NetX Duo TFTP Common Instance

ISDE Property	Value	Description
Maximum error string length (bytes)	64	Maximum error string length selection
Time to live	128	Time to live selection
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name

Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX/NetX Duo TFTP Client Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX/NetX Duo TFTP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin

TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.31.6 Using the NetX/NetX Duo TFTP Client Module in an Application

The following example assumes that a system is already establish with a working IP, ARP and UDP enabled and the link is running.

The steps in using the NetX/NetX Duo TFTP Client module in a typical application are:

1. Poll the nx_ip_status_check API for when the IP instance has a valid IP address.

2. To request a file write, call the nx_tftp_client_file_open API with NX_TFTP_OPEN_FOR_WRITE as the open_type input.

3. Allocate a packet using the nx_tftp_client_packet_allocate API and write the file data into the packet buffer. This packet is ready to send.

4. Send to the Server by calling the nx_tftp_client_file_write API.

5. Repeat until all file data is sent. Unless this API returns NX_TFTP_NOT_OPEN, the application should not release the packet.

6. Close the file by calling the nx_tftp_client_file_close.

7. To request a file read, call the nx_tftp_client_file_open API with NX_TFTP_OPEN_FOR_READ as the open_type input.

8. Receive file data by calling the nx_tftp_client_file_read API.

9. Continue until a packet of less than 512 bytes is received. Release the packets back to the packet pool when done with it.

10. Use the nx_tftp_client_file_close API to close the file.

11. Delete instance by nx_tftp_client_delete API when done with the TFTP Client.

The following figure illustrates common steps in a typical operational flow diagram:

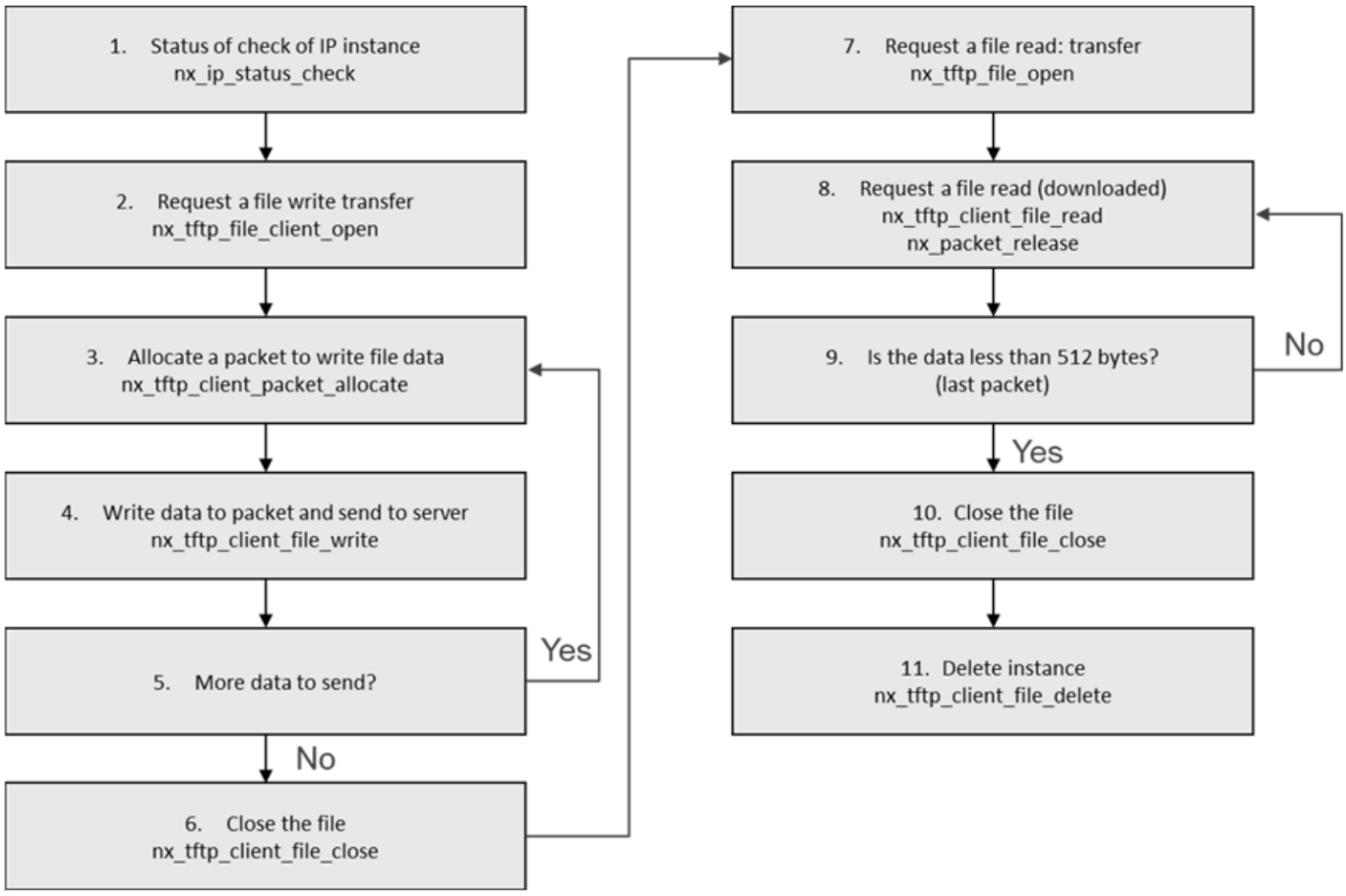


Figure 497: Flow Diagram of a Typical NetX/NetX Duo TFTP Client Module Application

4.3.32 NetX/NetX Duo TFTP Server

4.3.32.1 NetX and NetX Duo TFTP Server Introduction

The NetX™ Trivial File Transfer Protocol (TFTP) is a lightweight protocol designed for file transfers. Unlike more robust protocols, TFTP does not perform extensive error-checking and can have limited performance due to its stop-and-wait protocol. After a TFTP data packet is sent, the sender waits for an ACK to be returned by the recipient. Although this process is simple, it does limit the overall TFTP throughput. The TFTP package enables hosts to use the TFTP protocol over IP networks.

Note

Except for internal processing, the NetX Duo™ TFTP Server is identical to the NetX TFTP when performing the application, setup and running of a TFTP session.

NetX and NetX Duo TFTP Server Module Features

- NetX Server TFTP module is compliant with RFC 1350 and related RFCs.
- Provides high-level APIs for:
  - Creating and deleting a TFTP Server
  - Starting and stopping the TFTP Server task thread

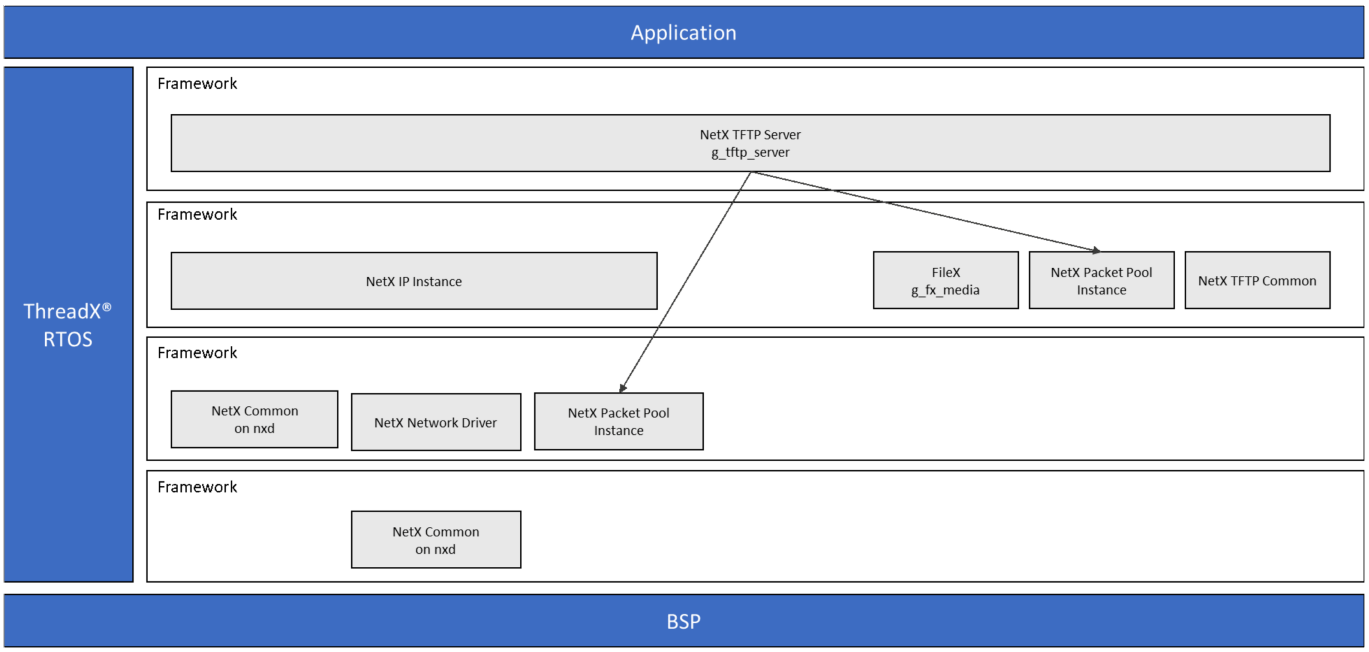


Figure 498: NetX/NetX Duo TFTP Server Module Block Diagram

Note

In the figure above, the FileX and NetX (or NetX Duo) Network Driver modules have multiple implementation options available. See the descriptions just after the module stack figure in Including the NetX and NetX Duo TFTP Server Module in an Application for additional details.

4.3.32.2 NetX and NetX Duo TFTP Server Module APIs Overview

The NetX TFTP Server module defines APIs for creating, deleting, generating response packets, response sending and getting information from a received packet. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX and NetX Duo TFTP Server Module API Summary

Function Name	Example API Call and Description
nx_tftp_server_create	nx_tftp_server_create(&my_server, "My TFTP Server", server_ip, &ram_disk, stack_ptr, 2048, pool_ptr); Create TFTP server (IPv4 only)
nxd_tftp_server_create**	nxd_tftp_server_create(&my_server, "My TFTP Server", &server_ip, &ram_disk, stack_ptr, 2048, pool_ptr); Create TFTP server (IPv4 and IPv6 supported).
nx_tftp_server_delete	nx_tftp_server_delete(&my_server); Delete TFTP server.
nxd_tftp_server_delete**	nxd_tftp_server_delete(&my_server); Delete TFTP server.

nx_tftp_server_start	nx_tftp_server_start(&my_server); Start the TFTP server.
nxd_tftp_server_start**	nxd_tftp_server_start(&my_server); Start the TFTP server.
nx_tftp_server_stop	nx_tftp_server_stop(&my_server); Stop the TFTP server.
nxd_tftp_server_stop**	nxd_tftp_server_stop(&my_server); Stop the TFTP server.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Azure RTOS User's Manual in the References section.

**This API is only available in NetX Duo TFTP Server. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide for the Renesas Synergy™ Platform*.

Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_TFTP_POOL_ERROR*	Packet pool payload is too small for the 512 bytes of TFTP data.
NX_PTR_ERROR*	Invalid pointer input
NX_CALLER_ERROR*	Invalid caller of this service

Note

Lower-level drivers may return common error codes. See SSP User's Manual API References for the associated module for a definition of all relevant status return values.

*These error codes are only returned if error checking is enabled. Please refer to the NetX Duo User Guide for the Renesas Synergy™ Platform for more details on error checking services in NetX Duo.

4.3.32.3 NetX and NetX Duo TFTP Server Module Operational Overview

To function properly, the TFTP Clients portion of the NetX Duo TFTP package requires an already-created IP instance.

Note

When a TFTP Server instance is added to the project, an IP Instance for the TFTP Server with an ARP and TCP enabled is automatically created.

The UDP must be enabled on that same IP instance. The client portion of the NetX Duo TFTP package has no further requirements.

The TFTP Server portion of the NetX TFTP package requires complete access to the UDP port 69 for handling all client TFTP requests; the TFTP Server is also designed for use with the FileX® embedded file system. If FileX is not available, the user may port portions of the FileX used to their own environment (details in later module guide sections).

File names should be in the format of the target file system. Filenames should be NULL terminated ASCII strings, with full path information, if necessary. There is no specified limit in the size of TFTP file names in the NetX Server TFTP implementation.

TFTP Messages

The TFTP has a simple mechanism for opening, reading, writing and closing files, with 2-4 bytes of the TFTP header underneath the UDP header. The definition of the TFTP file open messages has the following format:

A	B	C	D	Fstart	Fend	0		O	C	T	E	T	0

Where,

File Open protocol field

A	B	C	D	2 Byte Opcode field
0	0	0	1	Open for read
0	0	0	2	Open for write

F_{start} - F_{end}, File name

0, 1-byte NULL termination character.

OCTET, ASCII "OCTET" to specify binary transfer

The definition of the TFTP write, ACK and error messages are slightly different:

| A | B | C | D | W | X | Y | Z | Nstart | Nend | |---|---|---|---|---|---|---|---|---|---|---|

Where,

Protocol field for TFTP write

A	B	C	D	2 Byte Opcode field
0	0	0	3	Data packet
0	0	0	4	ACK for last read
0	0	0	5	Error condition

$\mathbf{N_{start}}$ - $\mathbf{N_{end}}$,, n-byte Data field

WXYZ, 2-byte Block Number field (1-n)

TFTP Communication

The data packet payload containing the file to upload or download is sent in 512 byte chunks until the last packet contains less than 512 bytes, where the packet containing fewer than 512 bytes signals the end of file. The general sequence of events is as follows:

TFTP Read File Requests:

1. The client issues an Open for Read request with the file name and waits for a reply from the server.
2. The server sends the first 512 bytes of the file or less if the file size is less than 512 bytes.
3. The client receives data, sends an ACK, and waits for the next packet from the server for files containing more than 512 bytes.
4. The sequence ends when the client receives a packet containing fewer than 512 bytes.

TFTP Write Requests:

1. The client issues an Open for Write request with the file name and waits for an ACK with a block number of 0 from the server.
2. When the server is ready to write the file, it sends an ACK with a block number of zero.
3. The client sends the first 512 bytes of the file (or less for files less than 512 bytes) to the server and waits for an ACK back.
4. The server sends an ACK after the bytes are written.
5. The sequence ends when the client completes writing a packet containing fewer than 512 bytes.

NetX and NetX Duo TFTP Server Module Important Operational Notes and Limitations

NetX and NetX Duo TFTP Server Module Operational Notes

The NetX TFTP Server module requires FileX media (Block media or USB Mass Storage). When a TFTP Server stack element is added to the project, an **Add FileX** box is attached to it. The configurator automatically sets up and initializes the FileX media for the server before the server is started. For details on configuring FileX, see the *FileX™ User's Guide* for the Renesas Synergy™ Platform.

The NetX TFTP Server also requires a packet pool for transmitting packets; it can share the IP default packet pool or create a separate packet pool. (Details on setting the TFTP Server packet pool are found in Including the NetX and NetX Duo TFTP Server Module in an Application.)

NetX and NetX Duo TFTP Server Module Limitations

- The TFTP Server maintains a TFTP Client session, even when the client stops responding, responses stop if the retransmission on client request support property is not enabled. In this manner, the TFTP Server can potentially fill up with dropped client connections and not be able to accept new client requests.
- The TFTP Server cannot respond to a duplicate client packet if retransmission on client request support property is not enabled. Duplicate packets are simply dropped and the TFTP Server does not send out any ACK or data packets, resulting in the client and the server being deadlocked.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.32.4 Including the NetX and NetX Duo TFTP Server Module in an Application

This section describes how to include either or both the NetX and NetX Duo TFTP Server module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX/NetX Duo TFTP Server module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX and NetX Duo TFTP Server Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_tftp0 NetX TFTPServer	Threads	New Stack> X-Ware> NetX> Protocols> NetX TFTPServer
g_tftp0 NetX Duo TFTPServer	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo TFTP Server

When the NetX and/or NetX Duo TFTP Server module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

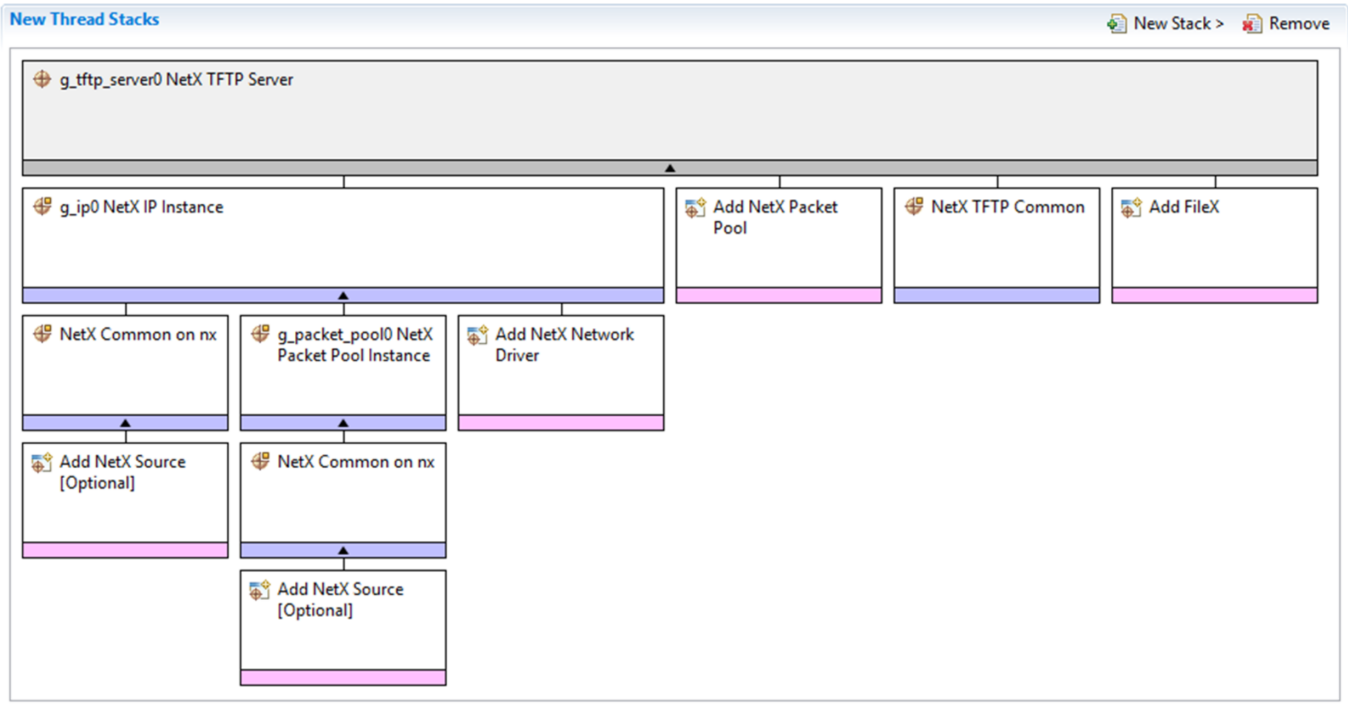


Figure 499: NetX and NetX Duo TFTP Server Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

Additionally, in the stack above, the FileX stack has also not been populated yet. There are multiple possible selections for the FileX module; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- FileX Stub
- FileX on Block Media (implemented on Block Media Framework on sf_block_media_ram)
- FileX on USB Mass Storage (implemented on USBX Host Class Mass Storage)

4.3.32.5 Configuring the NetX and NetX Duo TFTP Server Module

The NetX and NetX Duo TFTP Server module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX and NetX Duo TFTP Server Module

ISDE Property	Value	Description
FileX Support	Enable, Disable Default: Enable	FileX support selection
Retransmission on client request support	Enable, Disable Default: Disable	Retransmission on client request support selection
Internal thread priority	16	Internal thread priority selection
Maximum clients to serve simultaneously	10	Maximum clients to serve simultaneously selection
Time slice for internal thread	2	Time slice for internal thread selection
Client request activity timeout check interval (ticks)	20	Client request activity timeout (ticks) selection

Ack or data retransmission interval (ticks)	200	Ack or data retransmission interval (ticks) selection
Maximum retries for transmission without response	5	Maximum retries for transmission without response selection
Maximum retries for transmission with duplicate response	2	Maximum retries for transmission with duplicate response selection
Name	g_tftp_server0	Module name
Internal thread stack size (bytes)	2048	Internal thread stack size (bytes) selection
Name of generated initialization function	tftp_server_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC Addresses. The configurable properties for the lower-level stack modules are provided in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX and NetX Duo TFTP Server Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX and NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0,0,0,0	Default gateway address selection

**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization function
Link status change callback	NULL	Link status change callback selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

** Indicates properties that are only available in NetX Duo.

Configuration Settings for the NetX and NetX Duo TFTP Common Instance

ISDE Property	Value	Description
Maximum error string length (bytes)	64	Maximum error string length selection
Time to live	128	Time to live selection
Type of Service for UDP requests	Normal, Minimum delay, Maximum data, Maximum reliabililty, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX and NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX and NetX Duo TFTP Server Module Clock Configuration

The ETHERC peripheral module uses the PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator Clock tab prior to a build or by using the CGC interface at run-time.

NetX and NetX Duo TFTP Server Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin Selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.32.6 Using the NetX and NetX Duo TFTP Server Module in an Application

The steps in using the NetX and NetX Duo TFTP Server module in a typical application are:

1. Create the TFTP server using the nx_tftp_server_create API.

2. Prepare the FileX on Block media or USB mass storage API.

3. Start the TFTP Server using the nx_tftp_server_start API.

4. The TFTP Server now periodically checks for inactivity on active client connections.

5. All received packets are checked for being duplicate packets the server already received.

6. Receive a client open for read request (internal operation).

7. Check if the server can accommodate another client request (set by Maximum clients to server simultaneously property) (internal operation).

8. Download packets of file data in 512 chunks till the last packet (internal operation).

9. Close a file and delete the client request (internal operation).

The following figure illustrates common steps in a typical operational flow diagram:

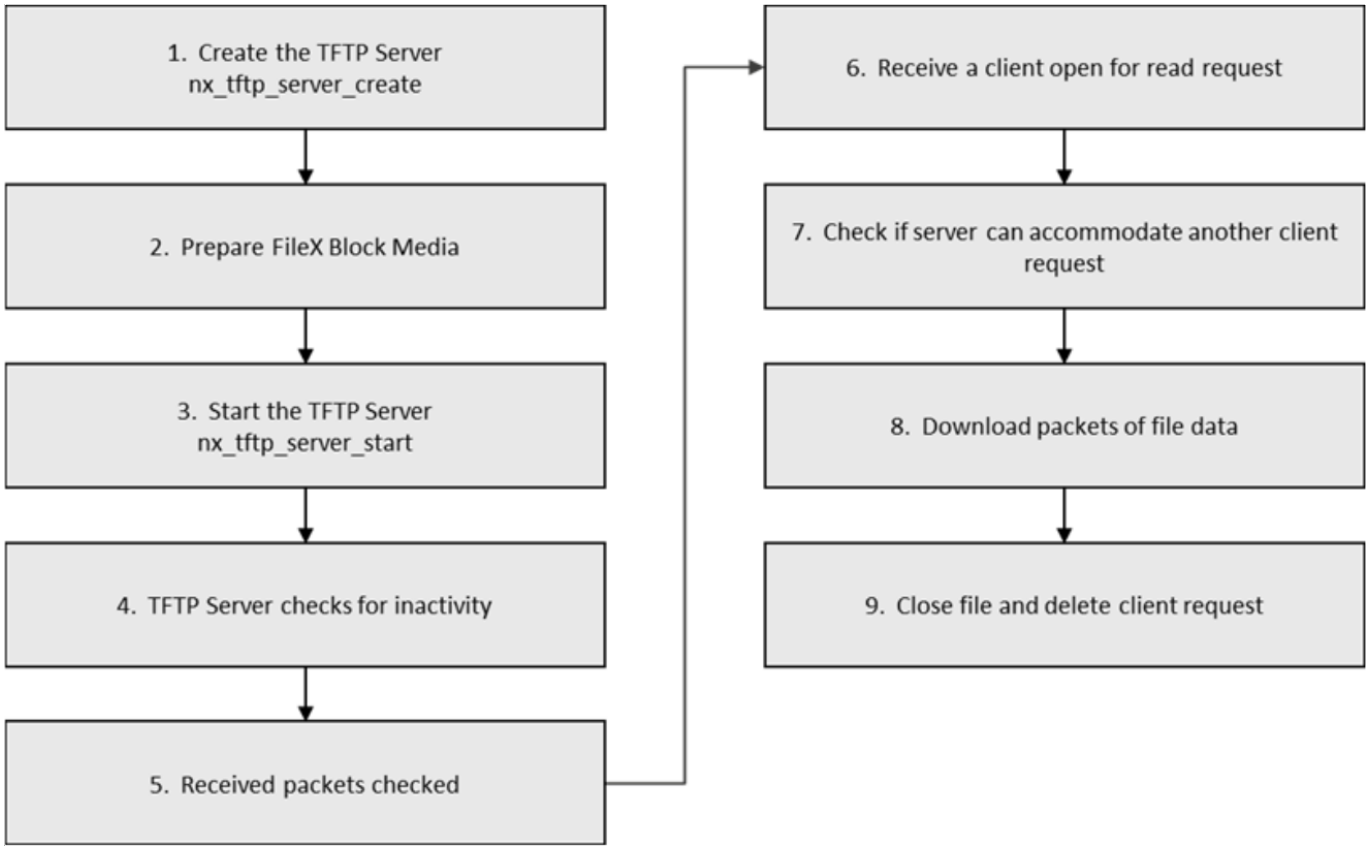


Figure 500: Flow Diagram of a Typical NetX and NetX Duo TFTP Server Module Application

4.3.33 NetX Duo MQTT Client

4.3.33.1 NetX Duo MQTT Client Introduction

The MQTT (Message Queue Telemetry Transport) is a communication protocol based on a publisher/subscriber model. A data producer can publish information to other clients through a broker. Multiple data consumers, if interested in a topic, can subscribe to the topic through the broker. The broker is responsible for authentication and authorization of the clients and delivering published messages to its topic subscribers. In this publisher/subscriber model, multiple clients may publish data with the same topic. A client will receive the messages it publishes if the client subscribes to the same topic.

NetX Duo MQTT Client Module Features

- Compliant with OASIS MQTT Version 3.1.1 Oct 29th, 2014. The specification can be found at: http://mqtt.org/
- Provides option to enable/disable TLS for secure communications using NetX Secure in SSP
- Supports QoS and provides the ability to choose the levels that can be selected while publishing the message
- Internally buffers and maintains queue of received messages
- Provides mechanism to register callback when new message is received.
- Provides mechanism to register callback when connection with the broker is terminated.

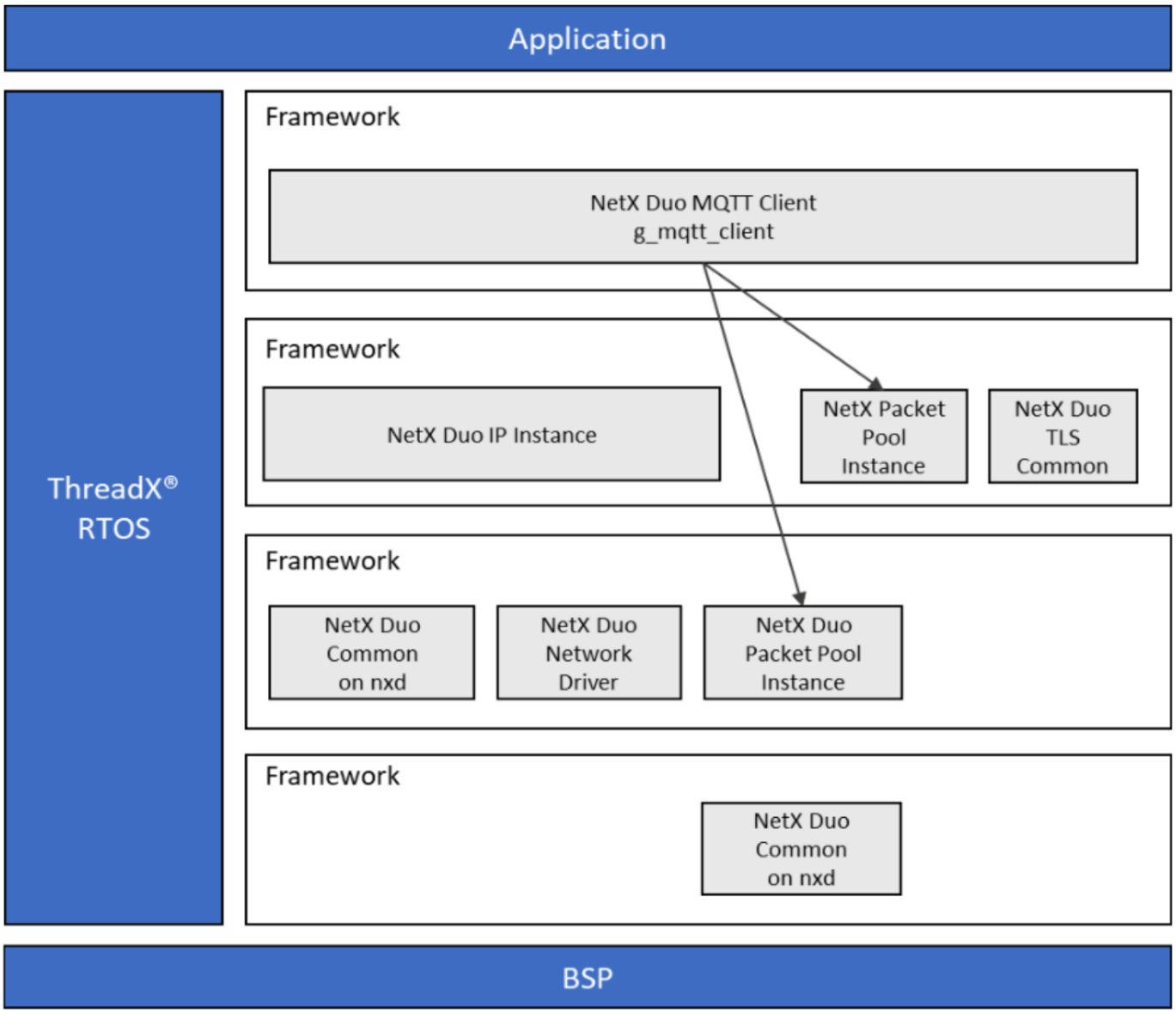


Figure 501: NetX Duo MQTT Client Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo MQTT Client Module in an Application for additional details.

4.3.33.2 NetX Duo MQTT Client Module APIs Overview

The NetX Duo MQTT Client Support module defines APIs for creating the MQTT Client, connecting to a broker, setting up TLS security and receiving MQTT messages. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo MQTT Client Module API Summary

Function Name	Example API Call and Description

nxd_mqtt_client_create	nxd_mqtt_client_create(&mqtt_client_secure, "my_client", CLIENT_ID_STRING, strlen(CLIENT_ID_STRING), ip_ptr, pool_ptr, (VOID*)mqtt_client_stack, sizeof(mqtt_client_stack), mqtt_thread_priority, (UCHAR*)client_memory, sizeof(client_memory)); Create an MQTT client with the specified Client ID, stack memory and stack size, and message block memory.
nxd_mqtt_client_connect	nxd_mqtt_client_connect(&mqtt_client, &server_ip, NXD_MQTT_PORT, MQTT_KEEP_ALIVE_TIMER, 0, NX_WAIT_FOREVER) Non-secure connect to the MQTT broker specifying broker IP address and port, keep alive timer, and disabling the clean session option
nxd_mqtt_client_secure_connect	nxd_mqtt_client_secure_connect(&mqtt_client_secure, &server_ip, NXD_MQTT_TLS_PORT, tls_setup_amazon, 600, 1, NX_WAIT_FOREVER) Connect to broker with TLS security using the tls_setup_amazon, which is a user-defined function, to set up TLS and set TLS parameters. The clean session option is enabled. This is only available if the NetX Duo library is built with NX_SECURE_ENABLE set, and if the MQTT client property NX Secure is set.
nxd_mqtt_client_login_set	nxd_mqtt_client_login_set(mqtt_client_ptr, "Username", strlen("Username"), "Password", strlen("Password"); Set the optional MQTT username and password. This must be called before the nxd_mqtt_client_connect or nxd_mqtt_client_secure_connect call if the broker requires username and password.
nxd_mqtt_client_message_get	nxd_mqtt_client_message_get(&mqtt_client_secure, &topic, &topic_length, &message, &message_length, &packet_ptr); Retreive a published MQTT message for the specified topic.
nxd_mqtt_client_receive_notify_set	nxd_mqtt_client_receive_notify_set(&mqtt_client_secure, my_notify_func); Specify the function the MQTT Client thread task calls when an MQTT message is received.
nxd_mqtt_client_subscribe	nxd_mqtt_client_subscribe(&mqtt_client_secure, TEST_SUBSCRIBE_TOPIC_NAME, strlen(TEST_SUBSCRIBE_TOPIC_NAME), 0); Send a subscriber message to the broker for the specified topic for QoS (quality of service) level 0.

nxd_mqtt_client_unsubscribe	nxd_mqtt_client_unsubscribe(NXD_MQTT_CLIENT *mqtt_client_pr,CHAR *topic_name, UINT topic_name_length); Send an unsubscriber message to the broker for the specified topic.
nxd_mqtt_client_publish	nxd_mqtt_client_publish(&mqtt_client_secure, TEST_SUBSCRIBE_TOPIC_NAME, strlen(TEST_SUBSCRIBE_TOPIC_NAME), message_buffer, strlen(message_buffer), 0, 1, NX_WAIT_FOREVER); Send a message to the broker for the specified topic previously subscribed to for QoS (quality of service) level 1, and the retain message option disabled.
nxd_mqtt_client_disconnect	nxd_mqtt_client_disconnect(&mqtt_client); Disconnect from the MQTT broker.
nxd_mqtt_client_disconnect_notify_set	nxd_mqtt_client_disconnect_notify_set(mqtt_client_ptr, my_disconnect_notify); Specify the user defined function for the MQTT Client thread task to call if the broker initiates disconnecting from the client.
nxd_mqtt_client_delete	nxd_mqtt_client_delete(mqtt_client_ptr); Delete the MQTT instance, clear transmit and message queue messages
nxd_mqtt_client_will_message_set	nxd_mqtt_client_will_message_set(mqtt_client_ptr, will_topic, will_topic_length, "will_message", strlen("will_message"), 0, 1); Set the optional MQTT Client will message without the retain will message option, for QOS 1. If a will message is needed, this must be called before connecting to the broker.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section. Refer to the Azure RTOS NetX Duo MQTT Client User's Manual for additional information on MQTT Client Module API functions.

4.3.33.3 NetX Duo MQTT Client Module Operational Overview

The MQTT (Message Queue Telemetry Transport) is a protocol based on the NetX Duo TCP/IP stack and a publisher-subscriber model. A client can publish information to other clients through an MQTT Server (broker). A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for delivering published messages to its clients who subscribe to the topic. In this publisher-subscriber model, multiple clients may publish data with the same topic. A client will receive a message it publishes if the client subscribes to the same topic.

The following figure provides an overview of the MQTT Client publish/subscribe model:

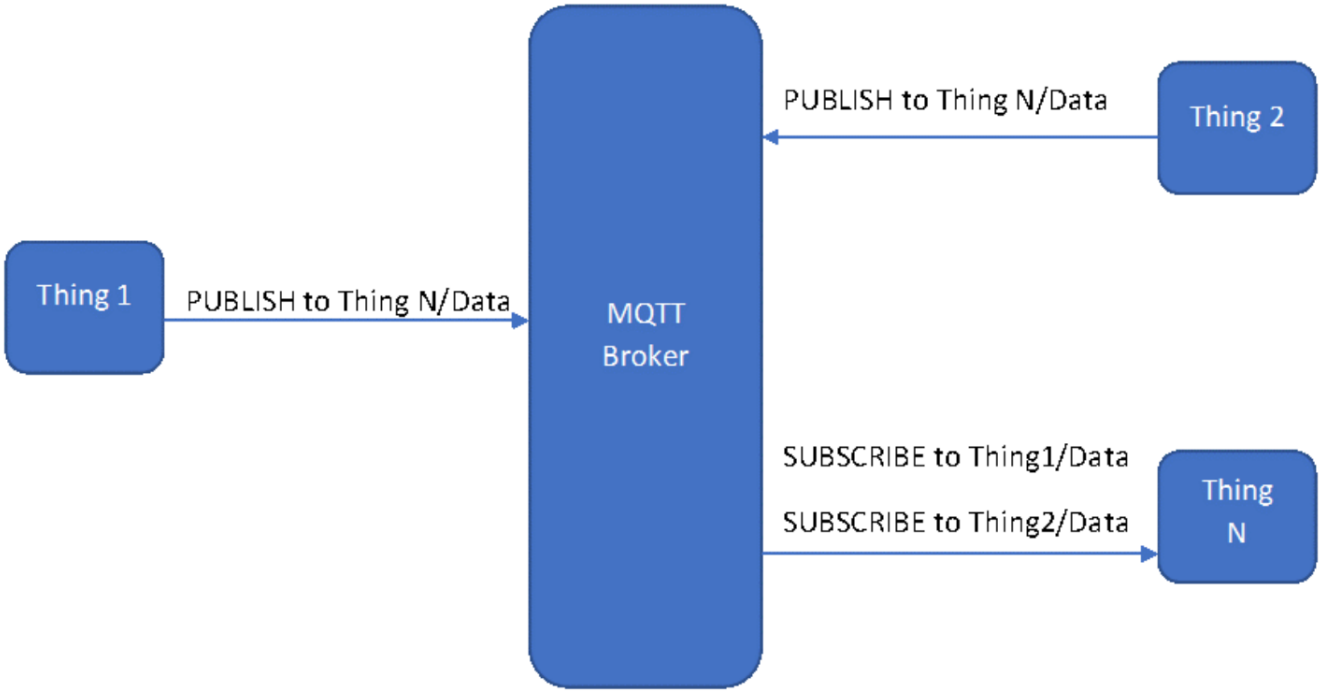


Figure 502: NetX Duo MQTT Client Module MQTT Client Publish/Subscribe Model

NetX Duo MQTT Client Module Publish/Subscribe Model

The NetX Duo MQTT client module can be used in the normal mode or the secure mode.

NetX Duo MQTT Client Module Normal Mode Operational Description

In normal mode, the communication between the MQTT client and broker is not secure.

NetX Duo MQTT Client Module Secure Mode Operational Description

In Secure mode, the communication between the MQTT client and broker is secured using the TLS protocol. In the thread pane, the TLS protocol is represented by "Add NetX Duo TLS common [Optional]" block.

Depending on the use case, a client may choose one of the 3 QoS levels when publishing a message:

QoS 0: The message is delivered at most once. Messages sent with QoS 0 may be lost.

QoS 1: The message is delivered at least once. Messages sent with QoS 1 may be delivered more than once.

QoS 2: The message is delivered exactly once. Messages sent with QoS 2 is guaranteed to be delivered, with no duplication.

Note

This implementation of MQTT client does not support QoS level 2 messages.

Since QoS 1 and QoS 2 are guaranteed to be delivered, the broker keeps track the state of QoS 1 and QoS 2 messages sent to each client. This is particularly important for clients that expect QoS1 or QoS 2 messages. The client may be disconnected from the broker (for example when the client reboots, or the communication link is temporarily lost). The broker must store QoS 1 and QoS 2 messages so the messages can be delivered later once the client is reconnected to the broker.

However, the client may choose not to receive any stale messages from the broker after reconnection. The client can do so by initiating the connection with the clean_session flag set to NX_TRUE (1) in the nxd_mqtt_client_connect API. In this case, upon receiving the MQTT CONNECT message, the broker shall discard any session information associated with this client, including undelivered or unconfirmed QoS 1 or QoS 2 messages.

If the clean_session flag is to NX_FALSE, the server shall resend the QoS 1 and QoS 2 messages. The MQTT Client also resends any un-acknowledged messages if clean_session is set to NX_TRUE. This acknowledgment is different from the TCP socket layer ACK, although that happens as well. The MQTT client sends an MQTT acknowledgment message to the broker upon receipt of a message, and gets one back when it publishes a message.

Incoming MQTT messages are stored in the receive queue of the MQTT client instance. The application retrieves these messages by calling the nxd_mqtt_client_message_get API which returns both the topic and the topic message. The application must ensure to provide a large enough buffer for each. The oldest message in the queue is returned to the caller first. The nxd_mqtt_client_message_get is non-blocking. If the MQTT client receive queue is empty, it returns immediately with an NXD_MQTT_NO_MESSAGE (0x1000A) status. This should not be handled as an error, but that the receive queue is empty.

To avoid polling the receive queue for incoming messages, the application can register a receive message callback function with the MQTT client by calling the nxd_mqtt_client_recieve_notify_set API. The callback function is defined as:

VOID (*receive_notify_callback)(NXD_MQTT_CLIENT *client_ptr, UINT message_count);

As the MQTT client receives messages from the broker, it invokes the callback function if the function is set. The callback function passes the pointer to the client control block and a message count value. The message count value indicates the number of MQTT messages in the receive queue. Note that this callback function executes in the MQTT client thread context. Therefore, the callback function should not execute any procedures that may block the MQTT client thread. The callback function should trigger the application thread to call the nxd_mqtt_client_message_get API to retrieve the messages. This is demonstrated in the module guide project.

To disconnect the MQTT client service, the application shall use the service nxd_mqtt_client_disconnect and nxd_mqtt_client_delete, APIs respectively. Calling nxd_mqtt_client_disconnect disconnects the TCP connection to the broker. It releases messages already received and stored in the receive queue. However, it does not release QoS level 1 messages in the transmit queue. QoS level 1 messages are retransmitted upon connection, assuming the clean_session flag is set to NX_FALSE.

The broker may initiate the disconnect from the client. The application can be notified of the disconnect request by registering a disconnect notify function with the MQTT Client. This is done by calling the nxd_mqtt_client_disconnect_notify_set API.

To delete an MQTT Client, call the nxd_mqtt_client_delete API. This releases all message blocks in the transmit queue and the receive queue. Unacknowledged QoS level 1 messages are also deleted.

Using Secure Communication

To secure the communication between MQTT client and broker, TLS protocol is required. In the thread pane, TLS protocol is represented by "Add NetX Duo TLS common \[Optional]" block. Adding NetX Duo TLS Common block enables the TLS support.

MQTT with TLS/NetX Duo Secure

When using TLS with MQTT Client, it is strongly recommended that the TLS setup callback in the nxd_mqtt_client_secure_connect call contain all of the TLS set up, including creating the TLS instance, defining the local certificates, allocating memory for remote certificate processing, and optional callbacks such as timestamp and certificate authentication. Once the callback has completed its operation successfully it should return Success. If the TLS setup callback returns a failure, the nxd_mqtt_client_secure_connect API also returns a failure to the application program.

Regardless if the MQTT Client was able to connect via TCP successfully or not, or whether the TLS session was successfully started, the application MUST call nxd_mqtt_client_disconnect to properly clear and reset the TLS session before attempting to reconnect again.

If the session was terminated improperly, nxd_mqtt_client_disconnect must still be called for the same reason.

For SSP 1.3.x, the TLS setup callback will need to call a memset on the NXD_SECURE_TLS_SESSION data block before (re)creating a TLS session (nxd_secure_tls_session_create).

The definition of the nxd_mqtt_client_secure_connect API with the TLS setup input is:

```
UINT nxd_mqtt_client_secure_connect(NXD_MQTT_CLIENT *client_ptr,
            NXD_ADDRESS *server_ip,
                                    UINT server_port,
                        UINT (*tls_setup)(
     NXD_MQTT_CLIENT *client_ptr,
     NX_SECURE_TLS_SESSION *session_ptr,
                                         NX_SECURE_X509_CERT
*,
                                         NX_SECURE_X509_CERT *),
                                    UINT keepalive,
                                    UINT clean_session,
                                    ULONG wait_option)
```

Add this logic to the tls_setup callback function (assuming the MQTT Client instance name is g_mqtt_client0):

```
    session_ptr = &(g_mqtt_client0.nxd_mqtt_tls_session);
    memset(session_ptr, 0, sizeof(NX_SECURE_TLS_SESSION));
```

```
status = nxd_secure_tls_session_create(....)
```

If memset is not called, the TLS nxd_secure_tls_session_create call may not succeed. In SSP 1.4.0 it will no longer be necessary to call memset, but it is still strongly recommended to put all TLS setup, including TLS creation in the callback. It may seem wasteful to completely delete and recreate a TLS session. But the manner in which TLS is integrated into MQTT Client makes this the most sensible and reliable method to guarantee successful reconnection attempts.

For more details about TLS protocol, please see the NetX Duo TLS Secure Module Guide.

Multiple Instances of MQTT Client Per Device

For SSP 1.4.0 and earlier, a device cannot safely run multiple instances of the MQTT Client because the MQTT Client in these releases assumes global variables. That should be remedied in a subsequent release.

NetX Duo MQTT Client Module Important Operational Notes and Limitations

NetX Duo MQTT Client Module Operational Notes

The NetX Duo MQTT Client component is added by clicking on the (+) sign in the thread pane window -> Azure RTOS -> NetX Duo -> Protocols -> NetX Duo MQTT Client.

Adding the NetX Duo MQTT Client component to a project automatically adds the option to add the NetX Duo TLS component required for secure MQTT.

The MQTT Client properties are listed in the following table:

Property	Value
◢ Common	
Parameter Checking	Default (BSP)
NX Secure	Enable
Topic Name Max Length	12
Message Max Length	32
Keepalive Timer Rate (s)	1
Ping Timeout Delay (s)	1
Socket Timeout (in timer ticks)	0xFFFFFFFF
◢ Module g_mqtt_client_ethr NetX Duo MQTT Client	
Name	g_mqtt_client_ethr
Client ID Callback	mqtt_client_ethr_id_callback
Client ID Max Length	75
Client Thread Stack Size	4096
Number of Messages to be stored in memory	1
Client thread priority	2
Name of generated initialization function	mqtt_client_init_ethr
Auto Initialization	Enable

Figure 503: NetX Duo MQTT Client Module MQTT Client Block Configurable Properties

In the figure above, "Common" properties are those configurable options in the NetX Duo MQTT Client that are common to all instances of the MQTT client in the project. The "Module" properties are specific to each instance of MQTT Client in the project.

Common Properties

- NX Secure: This enables/disables TLS support. If this property is set to Enabled, the MQTT Client is built with TLS support. Note: enabling the property requires adding the NetX Duo TLS component to the project to supply the necessary source code to the project, or the project will not build. If set to Disabled, adding the NetX Duo TLS component has no effect

though the project will still build and run.
- Topic Name Max Length: The maximum topic length (in bytes) the application is going to subscribe to. The default is 12 bytes.
- Message Max Length: The maximum message length (in bytes) the application is going to send or receive. The default is 32 bytes.
- Keepalive Timer Rate: This timer is used to keep track of the time since last MQTT control message was sent, and sends out an MQTT PINGREQ message before the keep-alive time expires. The default value is 1 second.
- Ping Timeout Delay: The time MQTT client waits for PINGRESP from the broker for after it sends out MQTT PINGREQ. The default value is 1 second.

Module Properties

- Name: Name of the MQTT client instance
- Name of generated initialization function: Name of initialization function which creates MQTT client instance. The default is the auto-generated function auto generated function mqtt_client_init0.
- Auto Initialization: Enable/disable call to initialization function. If disabled, the application thread entry function must obtain the Client ID and create the MQTT Client instance.
- Client ID Callback: Callback function provided by user for the MQTT Client thread task to obtain a unique client ID. If Auto Initialization is disabled, this and the Client ID length have no effect.
- Client ID Max Length: Maximum Length in bytes of the client ID.
- Client Thread Stack Size: MQTT Client thread stack size in bytes.
- Number of Messages to be stored in Memory: MQTT client uses memory area to store messages. The memory needed for MQTT client operation depends on the amount of data being sent or received. The minimal memory size is the size of a single MQTT_MESSAGE_BLOCK instance which is 60 bytes. The default value is 1 MQTT_MESSAGE_BLOCK or 60 bytes. However, this it not a good choice if there will be multiple messages received before the application can receive them. Transmitted messages cannot be released until thee TCP socket receives an ACK for the data, or if the QoS level is 1 or higher and the MQTT Client has received an ACK from the MQTT server. So the module guide project uses 6 message blocks. That number can probably be reduced to 3 or 4.
- Client Thread Priority: MQTT Client thread priority.
- Name of Generated Initialization function: Name of the function that will call the nxd_mqtt_client_create API. If Auto Initialization is disabled, this has no effect. If it is, Synergy will create this function automatically.
- Auto Initialization: This determines if the function specified in the Name of Generated Initialization function option is called. If set to Enable, it will invoke this function. Otherwise if set to Disable, the application must call the nxd_mqtt_client_create API before using any NetX Duo MQTT Client services.

Setting a Unique Client ID

As mentioned previously, an MQTT client instance is created using nxd_mqtt_client_create() API. If the MQTT Client application is letting the ISDE create the MQTT client, then it must define the Client ID Callback. This will be called before the ISDE calls nxd_mqtt_client_create internally with a defined Client ID string. The MQTT Client component allows you to set the Client ID Callback and Client ID Max Length in the list of MQTT Client properties (see above Module Properties).

The Client ID should be unique and is one of the parameters the MQTT broker uses to identify the client.

The prototype for Client ID callback is as follows:

```
void mqtt_client_id_callback(char * p_client_id, uint32_t * p_client_id_length);
```

p_client_id is a pointer to the Client ID to obtain, thus it is an output parameter which will be filled in by this callback function. p_client_id_length is a pointer to the length of the Client ID, thus it is an input and output parameter. This mechanism enables the Client ID to be determined at run time instead of at compile time.

If Client ID Callback is left empty in the properties pane of e^2 studio, a compiler error occurs. NULL is an acceptable entry. If your application prefers to create the MQTT Client directly, set this callback to NULL and set *Auto Initialization* to Disabled. Then when your application calls the nxd_mqtt_client_create API, provide the Client ID string directly, and the length of it as the input parameters:

```
    /* Create MQTT client instance. */
    nxd_mqtt_client_create(&mqtt_client, "my_client", CLIENT_ID_STRING,
strlen(CLIENT_ID_STRING), ip_ptr, pool_ptr, (VOID*)mqtt_client_stack, sizeof
(mqtt_client_stack), MQTT_THREAD_PRIORTY,     (UCHAR*)client_memory, sizeof
(client_memory));
```

Below is the sample reference implementation of the Client ID callback function which copies MAC address to a Client ID:

```
void mqtt_client_id_callback(char *p_client_id, uint32_t *p_client_id_length)
{
    uint32_t id_length;
    UCHAR mac_id[6]  = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06};
    if (*p_client_id_length < sizeof(mac_id))
    {
        id_length = *p_client_id_length;
    }
    else
    {
        id_length = sizeof(mac_id);
    }
     /* Copy MAC address to CLient ID and update client ID length */
     memcpy(p_client_id, mac_id, id_length);
    return;
      }
```

Note

It is possible to have an MQTT session with a zero length Client ID string. If an MQTT Client supplies a zero-byte Client ID, the Client MUST also set the clean_session input in the nxd_mqtt_client_connect API to NX_TRUE (1) as per the MQTT protocol. If the Client supplies a zero-byte Client ID with clean_session set to NX_FALSE (0), the Server will respond to the CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network Connection.

NetX Duo MQTT Client Module Limitations

- NetX Duo MQTT Client does not support sending or receiving QoS level 2 messages.
- NetX Duo MQTT Client does not support chained packets.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.33.4 Including the NetX Duo MQTT Client Module in an Application

This section describes how to include either or both the NetX Duo MQTT Client module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo MQTT Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo MQTT Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_mqtt_client0NetXDuo MQTT Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetXDuo MQTT Client

When the NetX Duo MQTT Client module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

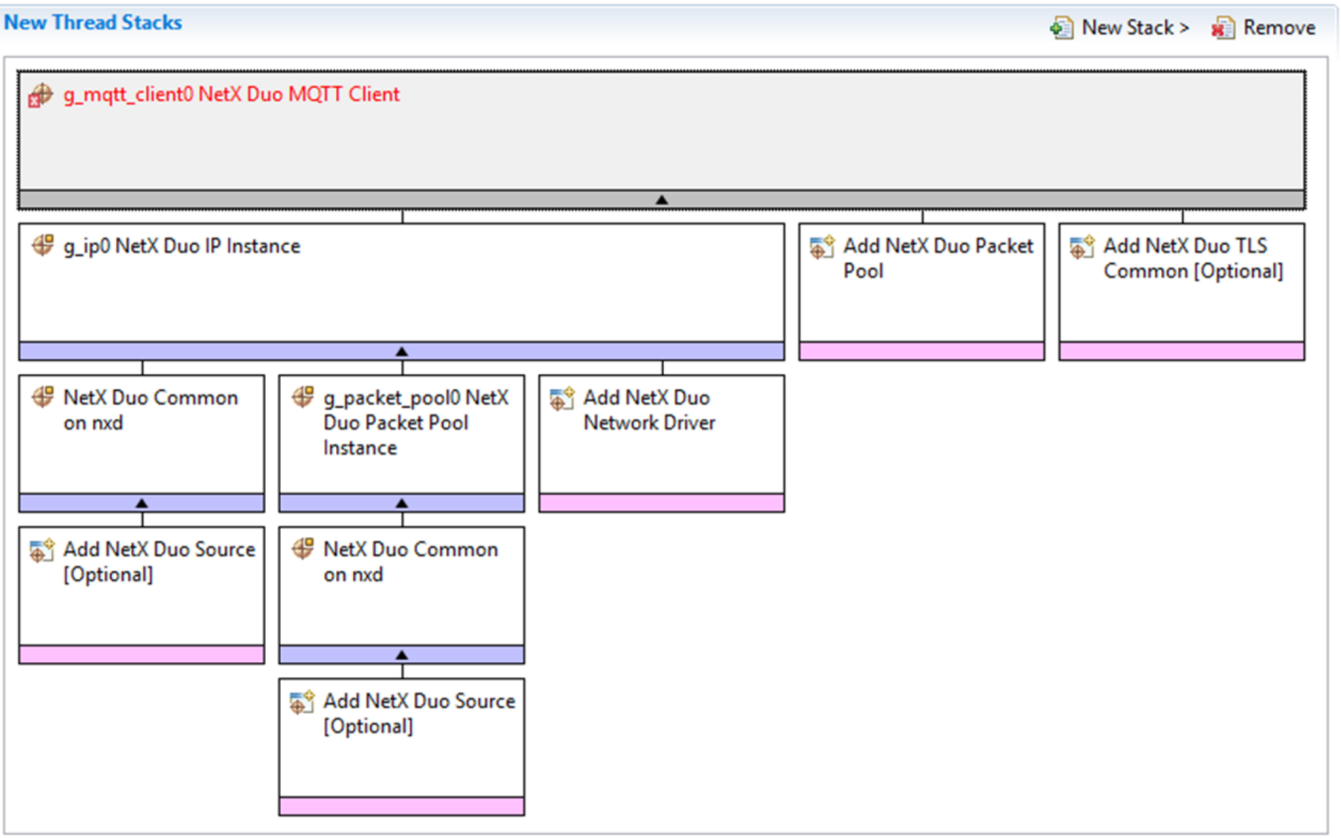


Figure 504: NetX Duo MQTT Client Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.33.5 Configuring the NetX Duo MQTT Client Module

The NetX Duo MQTT Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo MQTT Client Module

ISDE Property	Value	Description
NX Secure	Enable, Disable Default: Enable	This enables/disables TLS support. If this property is set to Enabled, the MQTT Client is built with TLS support. Note: enabling the property requires adding the NetX Duo TLS component to the project to supply the necessary source code to the project, or the project will not build. If set to Disabled, adding the NetX Duo TLS component has no effect though the project will still build and run.
Topic Name Max Length	12	The maximum topic length (in bytes) the application is going to subscribe to. The default is 12 bytes.
Message Max Length	32	The maximum message length (in bytes) the application is going to send or receive. The default is 32 bytes.
Keepalive Timer Rate(s)	1	This timer is used to keep track of the time since last MQTT control message was sent, and sends out an MQTT PINGREQ message before the keep-alive time expires. The default value is 1 second.
Ping Timeout Delay(s)	1	The time MQTT client waits for PINGRESP from the broker for after it sends out MQTT PINGREQ. The default value is 1 second.
Name	g_mqtt_client0	Name of the MQTT client instance.

Client ID Callback	mqtt_client_id_callback	Callback function provided by user for the MQTT Client thread task to obtain a unique client ID. If Auto Initialization is disabled, this and the Client ID length have no effect.
Client ID Max Length	12	Maximum Length in bytes of the client ID.
Client Thread Stack Size	4096	MQTT Client thread stack size in bytes.
Number of Messages to be stored in memory	1	MQTT client uses memory area to store messages. The memory needed for MQTT client operation depends on the amount of data being sent or received. The minimal memory size is the size of a single MQTT_MESSAGE_BLOCK instance which is 60 bytes. The default value is 1 MQTT_MESSAGE_BLOCK or 60 bytes. However, this it not a good choice if there will be multiple messages received before the application can receive them. Transmitted messages cannot be released until thee TCP socket receives an ACK for the data, or if the QoS level is 1 or higher and the MQTT Client has received an ACK from the MQTT server. So the module guide project uses 6 message blocks. That number can probably be reduced to 3 or 4.
Client thread priority	2	MQTT Client thread priority.
Name of generated initialization function	mqtt_client_init0	Name of the function that will call the nxd_mqtt_client_create API. If Auto Initialization is disabled, this has no effect. If it is, Synergy will create this function automatically

Auto Initialization	Enable, Disable Default: Enable	This determines if the function specified in the Name of Generated Initialization function option is called. If set to Enable, it will invoke this function. Otherwise if set to Disable, the application must call the nxd_mqtt_client_create API before using any NetX Duo MQTT Client services.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, you may require a longer Client ID string or to disable NX Secure. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo MQTT Client Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection

ARP cache storage units	Bytes, Entries Default: Bytes	ARP cache storage units selection
ARP Cache cache Size (in Bytes or storage units)	520	ARP Cache Size in Bytes/Entries selection. Must be a multiple of 52 Bytes. Note: 1 Entry = 52 Bytes
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1568	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo TLS Common

ISDE Property	Value	Description
Crypto Engine	Hardware	Crypto engine selection
Self Signed Certificates	Enable, Disable Default: Disable	Self signed certificates selection
PSK Cipher Suite	Enable, Disable Default: Disable	PSK cipher suite selection
ECC Cipher Suite	Enable, Disable Default: Disable	ECC cipher suite selection
X509 Strict Name Compare	Enable, Disable Default: Disable	X509 strict name compare selection
X509 Extended Distinguished Names	Enable, Disable Default: Disable	X509 extended distinguished names selection
Maximum RSA Modulus size (bits)	1024, 2048, 3072, 4096 Default: 4096	Maximum RSA modulus size (bits) selection
Server Mode	Enable, Disable Default: Enable	Server mode selection
Client Mode	Enable, Disable Default: Enable	Client mode selection

Name of generated initialization function	nx_secure_common_init	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Software Crypto

ISDE Property	Value	Description
Name	g_crypto_generic	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Crypto Hardware Accelerator

ISDE Property	Value	Description
Name	g_sf_el_nx_crypto	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SCE Common Driver

ISDE Property	Value	Description
Name	g_sce_0	Module name
Endian Flag	CRYPTO_WORD_ENDIAN_BIG, CRYPTO_WORD_ENDIAN_LITTLE Default: CRYPT_WORD_ENDIAN_BIG	Endian flag selection
Interfaces available from the InterfaceGet API		
AES<subset>	Enable, Disable Default: Enable	Enable or disable available interface (Plain-text ECB 128-bit, CBC 128-bit, etc. See configuration properties in the SSP Configurator for a complete list of those available for the target MCU)

RSA<subset>	Enable, Disable Default: Enable	Enable or disable available interface (Plain-text 1024-bit, 2048-bit, etc. See configuration properties in the SSP Configurator for a complete list of those available for the target MCU)
ECC<subset>	Enable, Disable Default: Enable	Enable or disable available interface (Plain-text 192-bit, 256-bit, etc. See configuration properties in the SSP Configurator for a complete list of those available for the target MCU)
HASH<subset>	Enable, Disable Default: Enable	Enable or disable available interface (SHA1, SHA224, etc. See configuration properties in the SSP Configurator for a complete list of those available for the target MCU)
True Random Number Generator	Enable, Disable Default: Enable	Enable or Disable True Random Number Generator interface

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo MQTT Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo MQTT Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.33.6 Using the NetX Duo MQTT Client Module in an Application

The steps in using the NetX Duo MQTT Client module in a typical application are:

1. Wait for the network link to be enabled by calling the nx_ip_status_check (or if your system has multiple network interfaces, call nx_ip_interface_status_check) with the NX_IP_LINK_ENABLED option.
2. Create an event flag group using the tx_event_flags_create API.
3. Connect to the MQTT server (broker) using the nxd_mqtt_client_connect API.
4. Set a receive notification callback using the nxd_mqtt_client_receive_notify_set API. The receive callback sets a flag when notified by the underlying NetX Duo socket services that it has received a packet on this connection.
5. Subscribe to a topic on the MQTT Server using the nxd_mqtt_client_subscribe API.
6. Publish a message to the topic using the nxd_mqtt_client_publish API.
7. Wait to receive messages by calling the tx_event_flags_get API.
8. Receive the message using the nxd_mqtt_client_message_get API. Note that unlike receiving packets from a socket, the MQTT Client need not be concerned about releasing packets. The MQTT Client thread task handles packet and message block allocate and release.
9. Unsubscribe from the topic by calling the nxd_mqtt_client_unsubscribe API, specifying the topic.
10. Disconnect from the topic by calling nxd_mqtt_client_disconnect API.

The following figure illustrates common steps in a typical operational flow diagram:

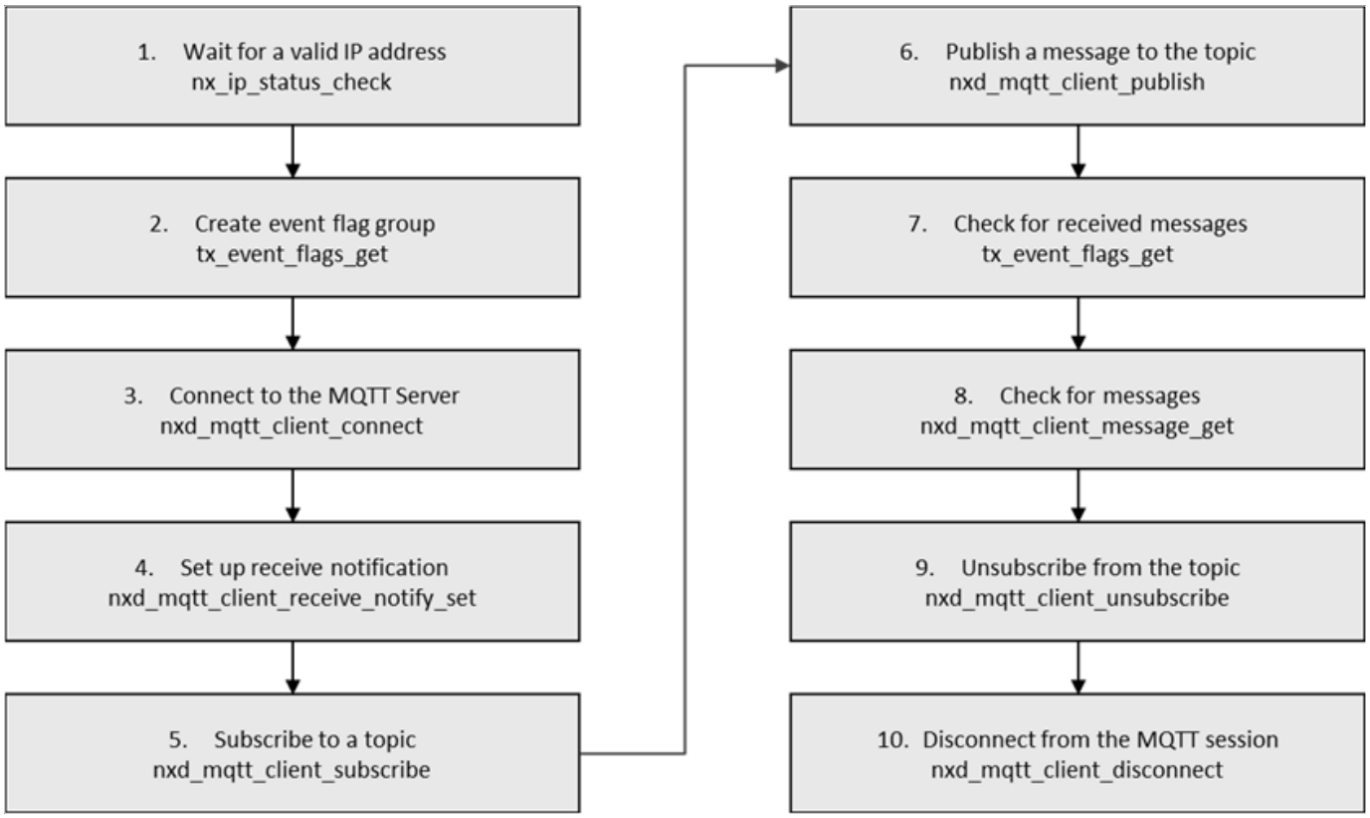


Figure 505: Flow Diagram of a Typical NetX Duo MQTT Client Module Application

4.3.34 NetX Duo NAT

4.3.34.1 NetX Duo NAT Introduction

The IP Network Address Translation (NAT) solves the problem of a limited number of Internet IPv4 addresses that arises when multiple devices need access to the Internet, but only one IPv4 Internet address is assigned by the Internet Service Provider (ISP). A NAT-enabled router is installed between the public and private network to translate between internal private IPv4 addresses and assigned public IPv4 address, so devices on the private network can share the same public IPv4 address.

NetX Duo NAT Module Features

- NetX NAT supports the following RFCs:
  - RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations
  - RFC 3022: Traditional IP Network Address Translator (Traditional NAT)
  - RFC 4787: Network Address Translation (NAT) Behavioral Requirements for Unicast User Datagram Protocol (UDP)
- NetX NAT provides the following high level APIs:
  - Creating and deleting a NAT server
  - Enabling and disabling NAT in NetX Duo

- Set callbacks for NAT to notify application if NAT entry table is full
- Creating static inbound entries in the NAT table

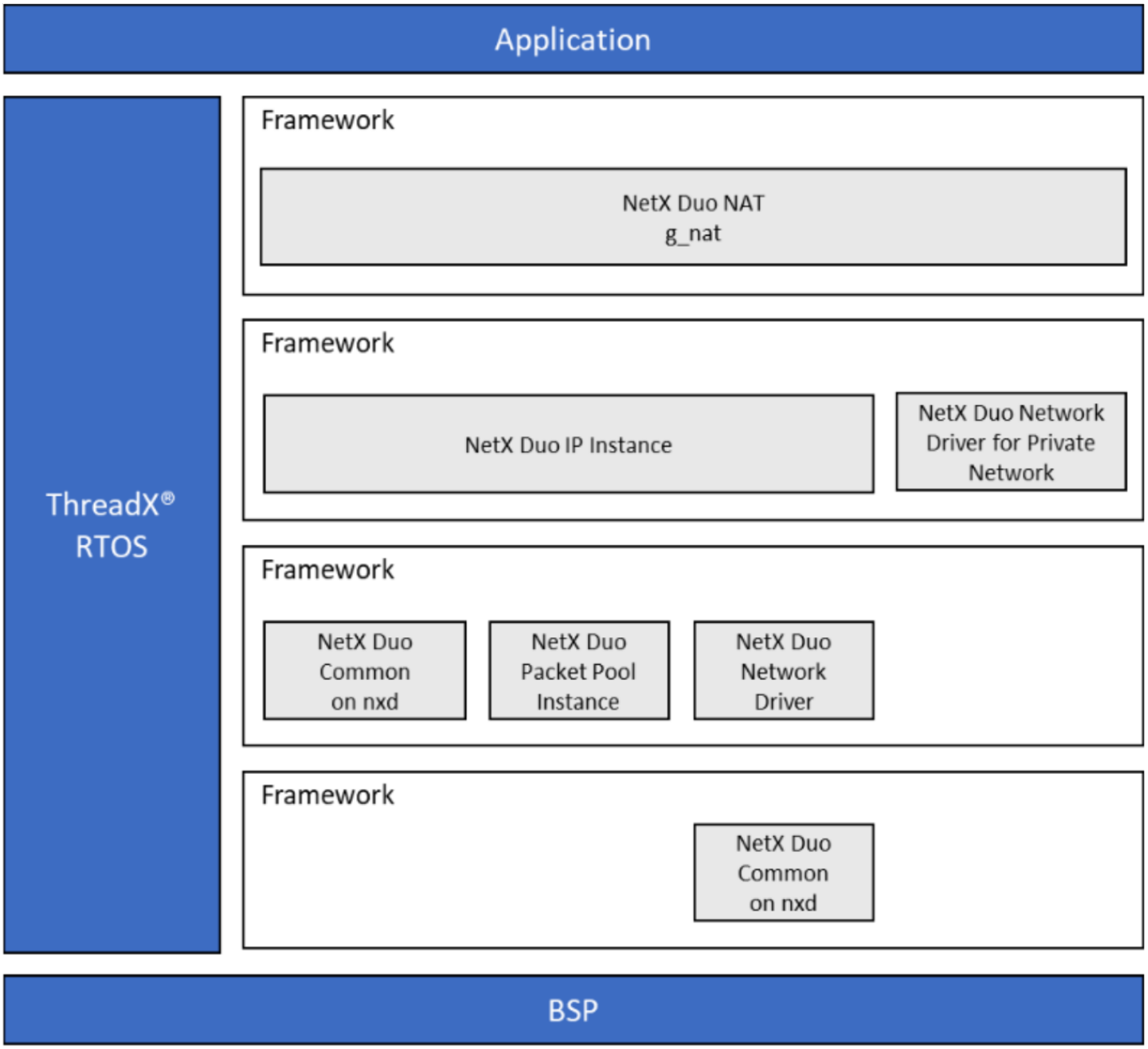


Figure 506: NetX Duo NAT Module Block Diagram

Note

In the figure above, The NetX Duo Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo NAT Module in an Application for additional details.

4.3.34.2 NetX Duo NAT Module APIs Overview

The NetX Duo NAT Module defines APIs for creating, deleting and enabling operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo NAT Module API Summary

Function Name	Example API Call and Description
nx_nat_create	nx_nat_create(nat_ptr, ip_ptr, global_interface_index, nat_cache, NX_NAT_ENTRY_CACHE_SIZE); Create a NAT Instance in which the network interface is the global interface (not the local/private network) with the specified network index.
nx_nat_delete	nx_nat_delete (nat_ptr); Delete a NAT instance.
nx_nat_enable	nx_nat_enable (nat_ptr); Enable the NAT server.
nx_nat_disable	nx_nat_disable (nat_ptr); Disable the NAT server.
nx_nat_cache_notify_set	nx_nat_cache_notify_set(nat_ptr, cache_full_notify_cb); Set the NAT cache full notify function to a user-defined notify function.
nx_nat_inbound_entry_create	nx_nat_inbound_entry_create(nat_ptr, entry_ptr, IP_ADDRESS(192,168,2,2), 5001, 5001, NX_PROTOCOL_TCP); Create an inbound translation table entry. This is typically used by application servers to allow clients to initiate a connection externally.
nx_nat_inbound_entry_delete	nx_nat_inbound_entry_delete(nat_ptr, delete_entry_ptr); Delete an inbound translation table entry.

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

Status Return Values

Name	Description
NX_SUCCESS	Successful NAT function
NX_PTR_ERROR*	Invalid input pointer parameter
NX_CALLER_ERROR*	Invalid caller (for example, must be a thread) of a service
NX_NAT_PARAM_ERROR*	Invalid non pointer input
NX_NAT_CACHE_ERROR*	Cache memory not 4 byte aligned, or is too small
NX_NAT_PORT_UNAVAILABLE	Invalid external port for creating static entry

NX_NAT_ENTRY_NOT_FOUND	Entry to delete is not found in Cache table
NX_NAT_ENTRY_TYPE_ERROR*	Invalid entry (not static) to delete

Note

Lower-level drivers may return common error codes. See the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

- These are error codes which are only returned if error checking is enabled. Refer to the *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User's Guide* for the Renesas Synergy™ Platform for more details on error-checking services in NetX and NetX Duo, respectively.

4.3.34.3 NetX Duo NAT Module Operational Overview

A NAT-enabled router typically has two network interfaces: one connected to the public Internet, the other connected to the private network. A typical router in this setup is responsible for routing IP datagrams between the private network and the public network based on the destination IP address. A NAT-enabled router performs address translation before routing an IPv4 datagram between the public and the private interface. Translation is established for each TCP or UDP session, based the internal source address and source port number, as well as the external destination address and destination port number. For the ICMP echo request and response datagram, the Internet Control Message Protocol (ICMP) query ID is used instead of the port number.

Typically, connections across the NAT boundary are initiated by the hosts on the private network sending outbound packets to an external host. In these cases, hosts are usually assigned dynamic (temporary) IP addresses. It is also possible to have connections initiated in the opposite direction if the private network has 'servers' (such as HTTP or FTP) that accept client requests from the external network. NAT applications typically assign these local hosts a static (permanent) IP address port.

The following three illustrations and accompanying steps illustrate the sequence of events when sending packets through a NAT router.

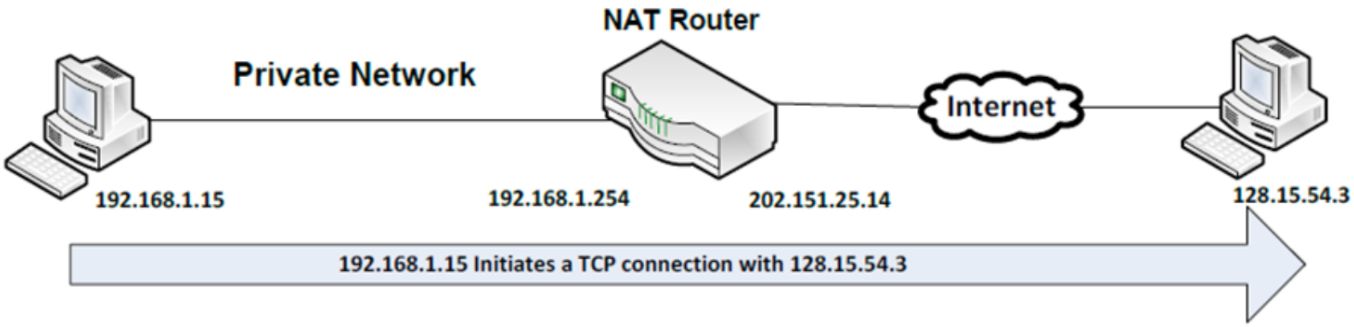


Figure 507: NetX Duo NAT Module Connection Start

Step 1. Client transmits a TCP SYN message to the web server. The Client address on the private network is 192.168.1.15, port number 6732; the destination address is 128.15.54.3, port number 80.

Step 2. The packet from the Client is received on the private network interface by the NAT router. The outbound traffic rule applies to the packet: the sender's (Client's) address is translated to the NAT router's public IP address 202.151.25.14, and sender (Client) source port number is translated to the TCP port number 2015 for transmission out on the public interface.

Step 3. The packet is then transmitted over the Internet and ultimately reaches its destination host 128.15.54.3. On the receiving side, notice in the following figure that the packet appears to have originated from 202.151.25.14, port number 2015 when it was in fact relayed from that IP address and port.

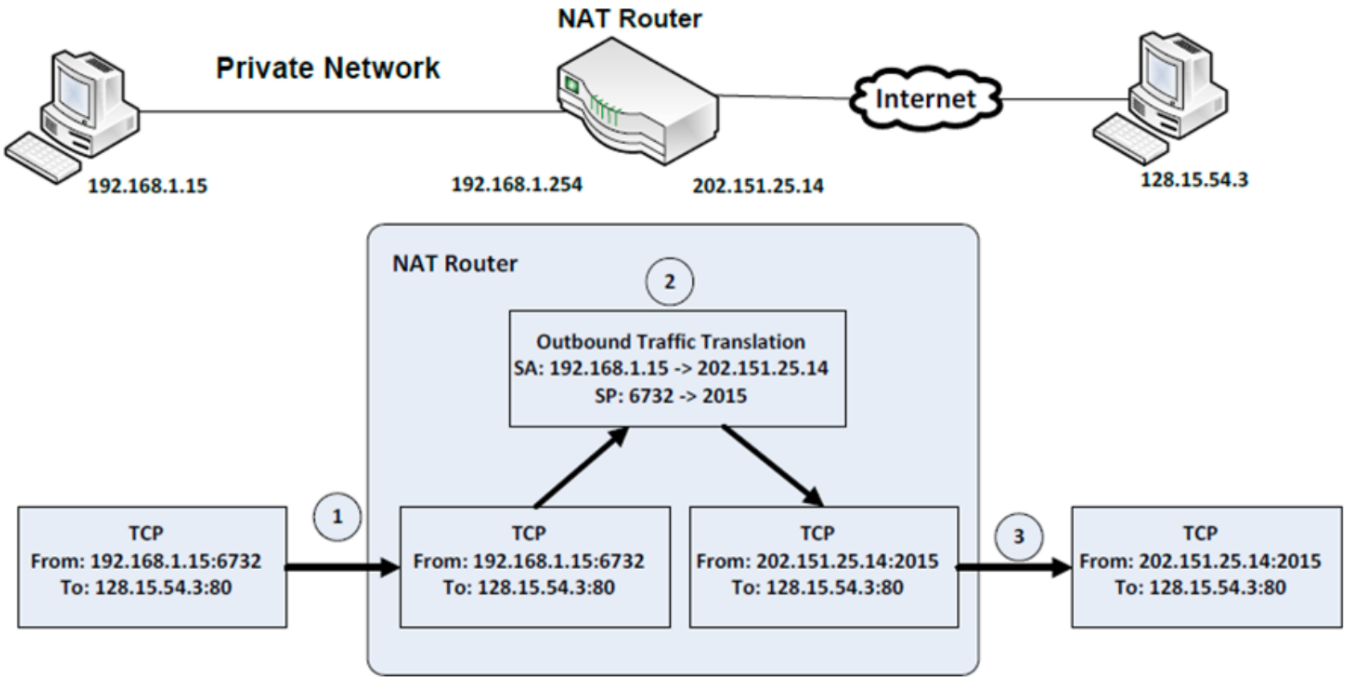


Figure 508: NetX Duo NAT Module Network Address Translation

Step 4. Host 128.15.54.3 sends back a response packet with the NAT router's Internet address as its destination.

Step 5. The packet reaches the NAT router. Since this is an in-bound packet, the in-bound translation rules apply: the destination address is changed back (translated) to the original sender's (Client's) IP address: 192.168.1.15, destination port number 6732.

Step 6. The packet is then forwarded to the Client through the interface connected to the internal network.

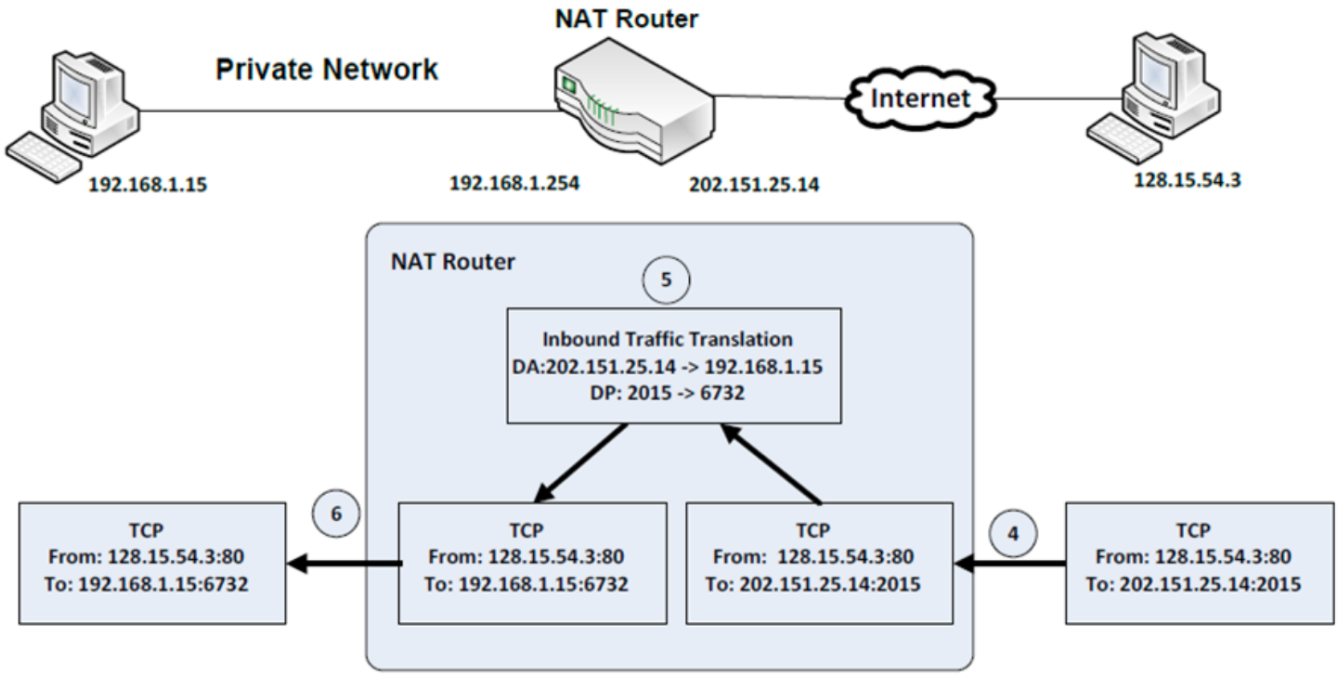


Figure 509: NetX Duo NAT Module Packet Return Path

When sending packets through a NAT router, the sender's Internet network address and port number is not exposed to other hosts on the public Internet.

To keep track of the network address translations for all active connections between local and external networks, the NetX Duo NAT-enabled router maintains a translation table with information about each private host connection that includes source and destination IP address and port number.

NetX Duo NAT is intended for use on an IPv4 router. For the NAT to work, the NetX Duo must be configured to forward-receive packets to an internal NetX Duo NAT handler. The handler determines whether the packet is received from the global network (inbound) or from the private network (outbound).

- For inbound packets, the handler determines whether it can forward (consume) the packet to the host on the private (local) network. To make the determination, the handler looks for a matching entry based on the packet destination address and port in the NAT translation table. If a matching entry is found, the handler translates the destination address to the matching private host IP address and port, and sends the packet to that host. If it cannot find an entry, it lets the NetX Duo process the packet as normal; as if the packet was intended for the NAT device itself.
- For outbound packets, the NAT handler checks the destination IP address to determine whether it can forward the packet out onto the global network or whether NetX Duo should handle the packet as it does normally. If the packet has a broadcast or loopback destination address, or an IP address which does not match the NAT device global network address, the NAT handler lets the NetX Duo handle the packet. Otherwise, the handler looks for a matching entry of the sender's IP and address in the translation table. If it finds a match, it translates the IP address and port to the local IP address and port, and forwards (consumes) the packet to the local host. If a previous entry is not found, the NAT creates an entry in the NAT translation table, translates the sender's IP address for the global interface and forwards the packet to the external host.

NetX Duo NAT Module Important Operational Notes and Limitations

NetX Duo NAT Module Operational Notes

- To enable the NAT, add the NetX Duo Source component to the Configurator pane, and set the *NAT* property of NetX Duo Source. The prebuilt NetX Duo source library does not have NAT enabled.
- The Maximum Physical Interfaces property, also in NetX Duo Source, must be set to 2, assuming one private network and one global network. Auto-generated code creates the IP instance using the global network IP address, and attach the secondary interface as the private network interface.
- There must be two network driver instances. The project for this module uses the NetX Port ETHER framework (sf_el_nx) for both interfaces but one network interface can be on Wi-Fi, or other network media. If using two sf_el_nx driver instances, ensure their names are not the same and that one references channel 0 and the other channel 1. The MAC addresses for a dual-ported driver are in a single NetX Port Ether configuration, so no need to change these but ensure they are not identical between Channel 0 and Channel 1.
- The function specified in the Name of the generated initialization function property must attach the secondary interface and create the NAT instance-if the Auto initialization property of the NAT instance is enabled in the configurator (by default it is enabled). The Private IPv4 Address property is the local IP address of the NAT device (server). The Global network interface index specifies the network interface the global network uses. By default, this index is set to zero (the primary interface of the IP instance), and the secondary interface is the local network (interface index 1).
- If *Auto Initialization* is disabled, then the NAT application must attach a secondary interface and create a NAT instance before using any NAT service. After attaching the secondary interface, the application should wait for the internal NetX Duo processing to enable the link on that interface using the nx_ip_interface_status_check API (see the module guide project example for how this is done).
- At runtime, the NetX Duo NAT Framework also creates a 4-byte aligned table, or cache, to store NAT translation entries. The size of the cache is set by the *Cache Size* property of the NAT instance. The default value is 1024 bytes (a NAT translation record is 28 bytes). The minimum size of a NetX Duo NAT Translation table is three entries. This value is set in the *Minimum count for translation entry* property which defaults to 3 but should be set to a larger number in a busy network with many local hosts.
- By default, entries created by the NAT server when receiving inbound or outbound packets are dynamic entries. They are assigned a timeout value (*Timeout for translation_entry* property); the default value of 240 seconds is the timeout recommended by RFC 2663. When an entry timeout expires, the entry is marked for deletion. However, because the NetX Duo NAT server implementation does not have a timer, it checks the entire table for expired entries and deletes them when adding a new entry to the table. If there are no entries that have expired, and the table is full, the NetX Duo NAT server notifies the application with the cache full callback. The application can set this callback with the nx_nat_cache_notify_set service.
- To create static entries that never expire, the application can use the nx_nat_inbound_entry_create service. These entries can only be created for inbound packets and are sometimes referred to as 'inbound rules.' The entries are intended for server applications, to allow clients to initiate connection sessions with the server. Internally, the NAT verifies whether the global and local ports requested in the inbound rule are available. To delete these entries, the application calls the nx_nat_inbound_entry_delete service.
- The NetX Duo NAT is configured with a range of TCP, UDP and ICMP translation ports to create unique local address: port entries for local hosts connecting with outside hosts.
    - TCP ports available for TCP translation ports are between the minimum value (*Minimum assigned port number for outbound TCP packets* property) and the

maximum value (*Maximum assigned port number for outbound TCP packets* property).
    - Similarly, for UDP entries ports, *Minimum assigned port number for outbound UDP packets* and *Maximum assigned port number for outbound UDP packets* properties.
    - ICMP packets do not have ports; instead, the ICMP ID can be used as the entry translation port, *Minimum ICMP query identifier* and *Maximum ICMP query identifier* properties.
- To start NetX Duo forwarding packets using the NAT protocol, the application calls the nx_nat_enable service. To suspend NetX forwarding, the application calls the nx_nat_disable service.
- Once the NAT is enabled, the NAT thread entry application lets the internal processing for NAT handle packets transmission between the global and local networks. It can create inbound rules at any time to allow a host on the global network to reach a local host, as is typically done for servers on the local network.

NetX Duo NAT Module Limitations

- Internet Group Management Protocol (IGMP) is not supported. NetX Duo NAT supports only TCP, UDP and ICMP.
- NAT protocol does not apply to IPv6 packet transmission.
- NetX Duo NAT does not include DNS or DHCP services, although NetX Duo NAT can integrate those services with its NAT operations.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.34.4 Including the NetX Duo NAT Module in an Application

This section describes how to include either or both the NetX Duo NAT module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo NAT module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo NAT Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_nat0NetX Duo NAT	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo NAT

When the NetX Duo NAT module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

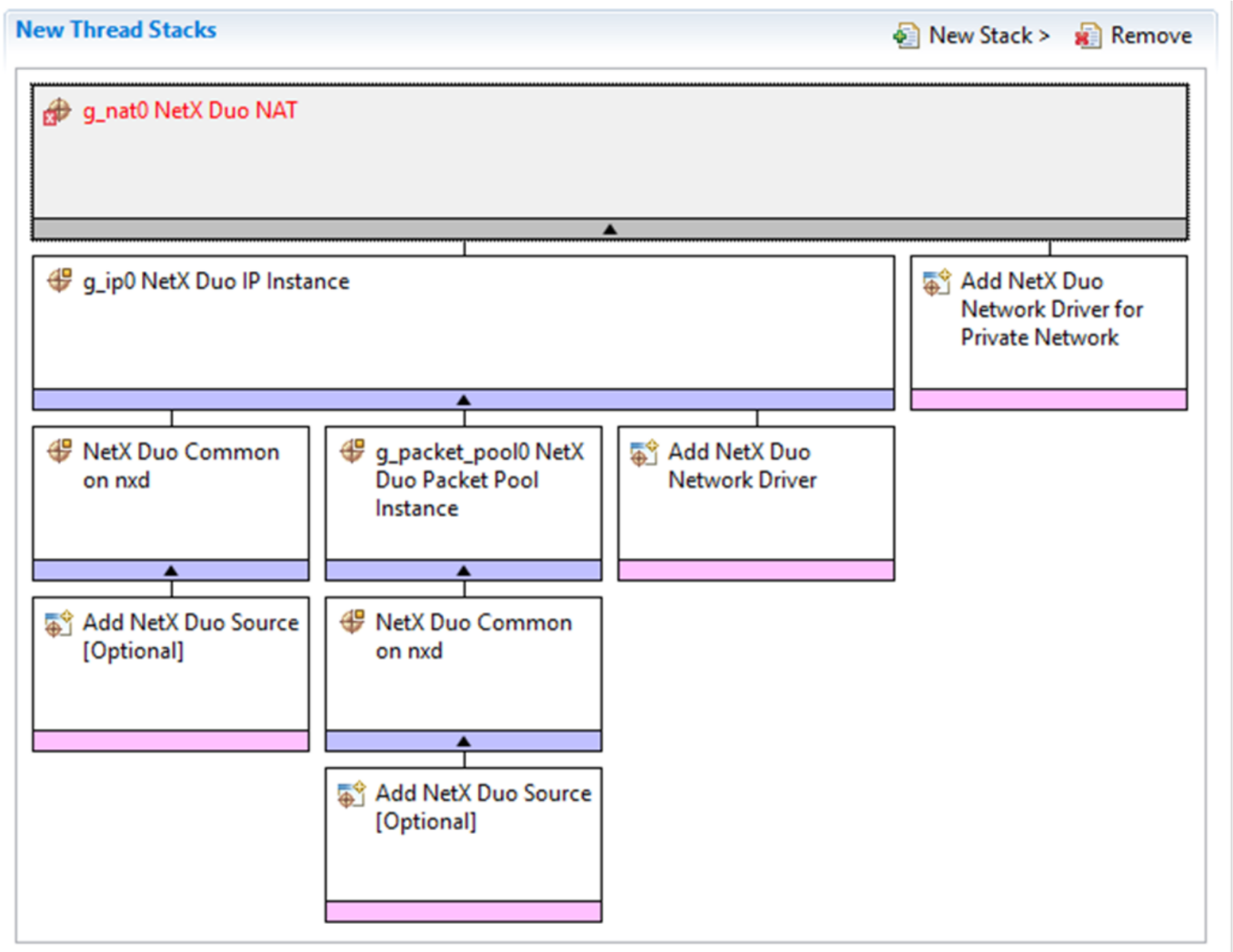


Figure 510: NetX Duo NAT Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are not all provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.34.5 Configuring the NetX Duo NAT Module

The NetX Duo NAT module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This

approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo NAT Module

ISDE Property	Value	Description
Minimum count for translation entry	3	Minimum count for translation entry selection
Timeout for translation entry (ticks)	240	Timeout for translation entry selection
Minimum assigned port number for outbound TCP packets	20000	Minimum assigned port number for outbound TCP packets selection
Maximum assigned port number for outbound TCP packets	30000	Maximum assigned port number for outbound TCP packets selection
Minimum assigned port number for outbound UDP packets	20000	Minimum assigned port number for outbound UDP packets selection
Maximum assigned port number for outbound UDP packets	30000	Maximum assigned port number for outbound UDP packets selection
Minimum ICMP query identifier	20000	Minimum ICMP query identifier selection
Maximum ICMP query identifier	30000	Maximum ICMP query identifier selection
Name	g_nat0	Module name
Cache size (bytes)	1024	Cache size selection
Private IPv4 Address (use commas for separation)	192,168,0,2	Private IPv4 Address selection
Private IPv4 Netmask (use commas for separation)	255, 255, 255, 0	Private IPv4 Netmask selection
Global network interface index	0	Global network interface index selection
Name of generated initialization function	nat_init0	Name of generated initialization function selection

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different IP addresses and subnet masks. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo NAT Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection

TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection

Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo NAT Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo NAT Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin

RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.34.6 Using the NetX Duo NAT Module in an Application

The following example assumes that a system is already established with a working IP, ARP, ICMP, TCP and UDP enabled, and that the link is running.

The steps in using the NetX Duo NAT module in a typical application are:

1. Wait for the IP instance to initialize the driver and have a valid IP address using the nx_ip_status_check API for the primary ("global") interface.
2. Wait for the IP secondary ("local") interface to also have a valid IP address at this point using the nx_ip_interface_status_check API.
3. Set the cache full callback to be notified when the NAT translation table it full by calling the nx_nat_cache_notify_set API [Optional].
4. Start packet forwarding by the IP layer in NetX as per NAT protocol by calling the nx_nat_enable API. The IP thread task and NAT services handle the rest.
5. Add static entries as desired (for example server applications that are waiting for client requests) by calling the nx_nat_inbound_entry_create [Optional]. Note this can be done before or after enabling NAT services.
6. Suspend NAT by calling the nx_nat_disable API.
7. Delete NAT by calling the nx_nat_delete API.

The following figure illustrates common steps in a typical operational flow diagram:

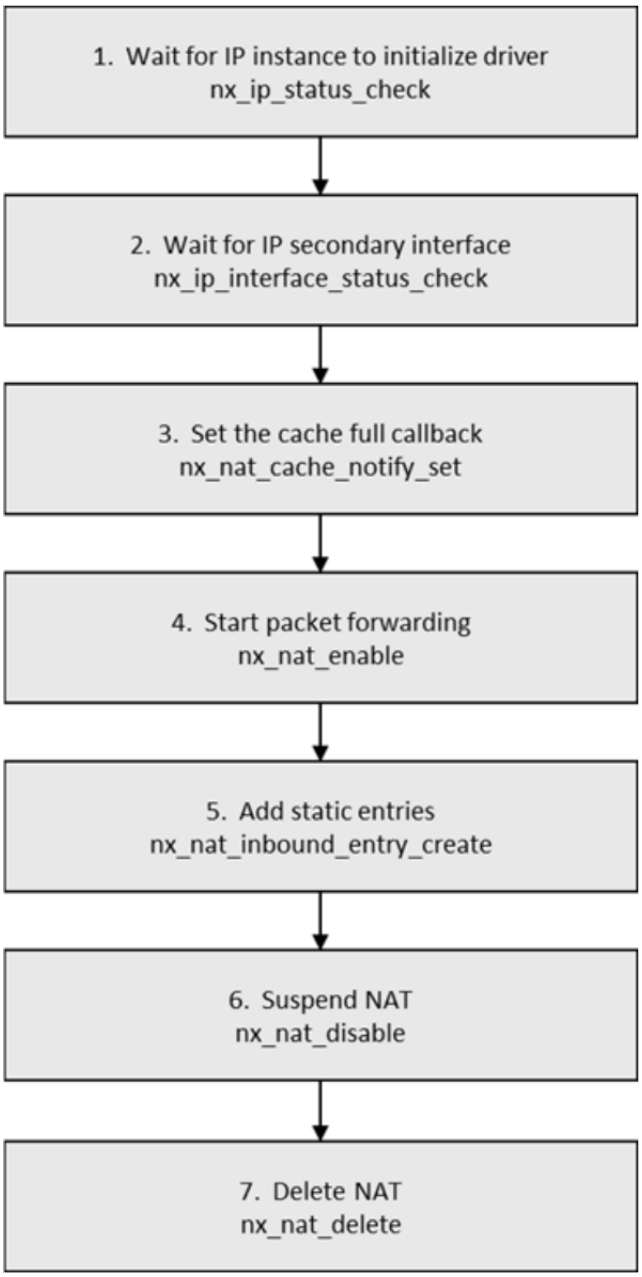


Figure 511: Flow Diagram of a Typical NetX Duo NAT Module Application

4.3.35 NetX Duo TLS Session

4.3.35.1 NetX Duo TLS Session Introduction

The NetX Duo TLS Session Module provides a high-level API for Transport Layer Security (TLS) protocol-based clients. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo TLS Session module is based on ThreadX "NetX Duo Secure," which implements the

Secure Socket Layer (SSL) and its replacement Transport Layer Security (TLS) protocol as described in RFCs 5246 (version 1.2) and 8446 (version 1.3). NetX Duo Secure also includes routines for basic X.509 (RFC 5280). NetX Duo Secure is intended for applications using the ThreadX RTOS.

The TLS/SSL protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

- Encryption: The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanism such as AES (Advanced Encryption Standard is used for data encryption.
- Authentication: A mechanism to check the peer's identity using certificates
- Integrity: A mechanism to detect message tampering and forgery to ensure that connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity

An application project that demonstrates the use of the TLS as part of an MQTT client is available on the Renesas web site. The "Synergy Enterprise Cloud Toolbox" application project and application note can be found by searching the Renesas web site for "R20AN0485."

NetX Duo TLS Session Module Features

- RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2
- RFC 8446 The Transport Layer Security (TLS) Protocol Version 1.3

    - *Note*

      TLS v1.0 and v1.1 are not supported by SSP. Users are strongly recommended to migrate their projects to TLS v1.2 or TLS v1.3 in SSP immediately.
- RFC 5280 X.509 PKI Certificates (v3)
- RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Transport Layer Security (TLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for TLS
- Supports TLS extensions for:
    - Secure Renegotiation Indication: This extension mitigates a Man-in-the-Middle attack vulnerability that could occur during a renegotiation handshake.
    - Server Name Indication: This extension allows a TLS Client to supply a specific DNS name to a TLS Server, allowing the server to select the correct credentials (assumes the server has multiple identity certificates and network entry points).
    - Signature Algorithms: This extension enables a TLS Client to provide a list of acceptable signature and hash algorithms to a TLS Server
- Supports X.509 extensions for:
    - Key Usage: Provides acceptable uses for a certificate's public key in a bitfield
    - Extended Key Usage: Provides additional acceptable uses for a certificate's public key using OIDs
- Certificate verification: Supports SHA-1, SHA-256, SHA-384,and SHA-512 signature hashing algorithms
- Subject Alternative Name: Provides alternative DNS names that are also represented by the certificate

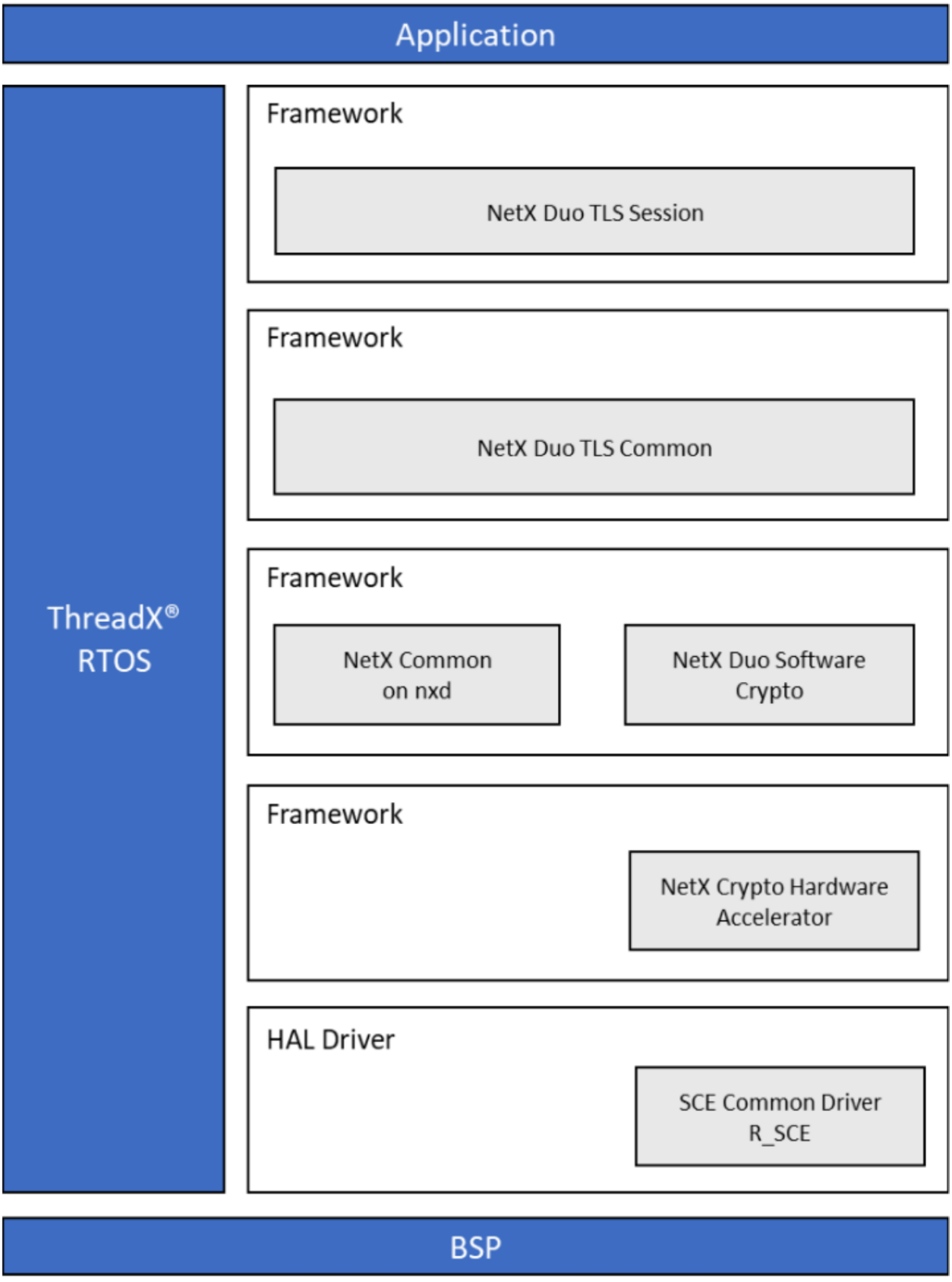


Figure 512: NetX Duo TLS Session Module Block Diagram

4.3.35.2 NetX Duo TLS Session Module APIs Overview

The NetX Duo TLS Support module defines APIs for creating and setting up a TLS security session. A complete list of the available APIs, an example API call and a short description of each can be found in the following table.

NetX Duo TLS Session Module API Summary

Function Name	Example API Call and Description
nx_secure_crypto_table_self_test	nx_secure_crypto_table_self_test(const NX_SECURE_TLS_CRYPTO *crypto_table,VOID *metadata, UINT metadata_size); Perform self-test on the crypto methods
nx_secure_module_hash_compute	nx_secure_module_hash_compute(NX_CRYPTO_METHOD *hamc_ptr,UINT start_address, UINT end_address, UCHAR *key, UINT key_length,VOID *metadata, UINT metadata_size,UCHAR *output_buffer, UINT output_buffer_size, UINT *actual_size); Computes hash value using user-supplied hash function
nx_secure_tls_active_certificate_set	nx_secure_tls_active_certificate_set(NX_SECURE_TLS_SESSION *tls_session,NX_SECURE_X509_CERT *certificate); Set the active identity certificate for a NetX Secure TLS Session
nx_secure_tls_initialize	nx_secure_tls_initialize(VOID); Initializes the NetX Secure TLS module
nx_secure_tls_remote_certificate_buffer_allocate	nx_secure_tls_remote_certificate_buffer_allocate (NX_SECURE_TLS_SESSION *session_ptr,UINT certs_number, VOID *certificate_buffer,ULONG buffer_size); Allocate space for all certificates provided by a remote TLS host
nx_secure_tls_remote_certificate_free_all	nx_secure_tls_remote_certificate_free_all(NX_SECURE_TLS_SESSION *session_ptr); Free space allocated for incoming certificates
nx_secure_tls_server_certificate_add	nx_secure_tls_server_certificate_add(NX_SECURE_TLS_SESSION *session_ptr, NX_SECURE_X509_CERT *certificate, UINT cert_id); Add a certificate specifically for TLS servers using a numeric identifier
nx_secure_tls_server_certificate_find	nx_secure_tls_server_certificate_find(NX_SECURE_TLS_SESSION *session_ptr, NX_SECURE_X509_CERT **certificate, UINT cert_id); Find a certificate using a numeric identifier

nx_secure_tls_server_certificate_remove	nx_secure_tls_server_certificate_remove(NX_SECURE_TLS_SESSION *session_ptr,UINT cert_id); Remove a local server certificate using a numeric identifier
nx_secure_tls_session_client_callback_set	nx_secure_tls_session_client_callback_set(NX_SECURE_TLS_SESSION *tls_session, ULONG (*func_ptr)(NX_SECURE_TLS_SESSION *tls_session, NX_SECURE_TLS_HELLO_EXTENSION *extensions, UINT num_extensions)); Set up a callback for TLS to invoke at the beginning of a TLS Client handshake
nx_secure_tls_session_x509_client_verify_configure	nx_secure_tls_session_x509_client_verify_configure(NX_SECURE_TLS_SESSION *session_ptr,UINT certs_number, VOID *certificate_buffer,ULONG buffer_size); Enable client X.509 verification and allocate space for client certificates
nx_secure_tls_session_renegotiate_callback_set	nx_secure_tls_session_renegotiate_callback_set(NX_SECURE_TLS_SESSION *tls_session, ULONG (*func_ptr)(struct NX_SECURE_TLS_SESSION_struct *session)); Assign a callback that will be invoked at the beginning of a session renegotiation
nx_secure_tls_session_renegotiate	nx_secure_tls_session_renegotiate(NX_SECURE_TLS_SESSION *tls_session, UINT wait_option) Initiate a session renegotiation handshake with the remote host
nx_secure_tls_session_server_callback_set	nx_secure_tls_session_server_callback_set(NX_SECURE_TLS_SESSION *tls_session, ULONG (*func_ptr)(NX_SECURE_TLS_SESSION *tls_session, NX_SECURE_TLS_HELLO_EXTENSION *extensions, UINT num_extensions)); Set up a callback for TLS to invoke at the beginning of a TLS Server handshake
nx_secure_tls_session_sni_extension_parse	nx_secure_tls_session_sni_extension_parse(NX_SECURE_TLS_SESSION *session_ptr, NX_SECURE_TLS_HELLO_EXTENSION *extensions,UINT num_extensions, NX_SECURE_X509_DNS_NAME *dns_name); Parse a Server Name Indication (SNI) extension received from a TLS Client
nx_secure_tls_session_sni_extension_set	nx_secure_tls_session_sni_extension_set(NX_SECURE_TLS_SESSION *session_ptr, NX_SECURE_X509_DNS_NAME *dns_name); Set a Server Name Indication (SNI) extension DNS name to send to a remote Server

nx_secure_tls_timestamp_function_set	nx_secure_tls_timestamp_function_set(NX_SECURE_TLS_SESSION *session_ptr,ULONG (*time_func_ptr)(void)); Assign a timestamp function to a NetX Secure TLS Session
nx_secure_x509_dns_name_initialize	nx_secure_x509_dns_name_initialize(NX_SECURE_X509_DNS_NAME*dns_name, const UCHAR *name_string, UINT length); Initialize an X.509 DNS name structure
nx_secure_x509_extended_key_usage_extension_parse	nx_secure_x509_extended_key_usage_extension_parse(NX_SECURE_X509_CERT*certificate, UINT key_usage); Find and parse an X.509 extended key usage extension in an X.509 certificate
nx_secure_x509_extension_find	nx_secure_x509_extension_find(NX_SECURE_X509_CERT*certificate, NX_SECURE_X509_EXTENSION *extension,USHORTextension_id); Find and return an X.509 extension in an X.509 certificate
nx_secure_x509_key_usage_extension_parse	nx_secure_x509_key_usage_extension_parse(NX_SECURE_X509_CERT*certificate, USHORT *bitfield); Find and parse an X.509 Key Usage extension in an X.509 certificate
nx_secure_tls_local_certificate_add	nx_secure_tls_local_certificate_add (tls_session, certificate); Adds an initialized certificate to a TLS session for use as a local identification certificate - the TLS Server certificate for TLS servers, and the Client certificate for TLS clients.
nx_secure_tls_local_certificate_remove	nx_secure_tls_local_certificate_remove(tls_session, common_name, common_name_length); Removes a certificate instance from the local certificates list, keyed on the Common Name field.
nx_secure_tls_metadata_size_calculate	nx_secure_tls_metadata_size_calculate(cipher_table, metadata_size); Determines the size of the buffer needed by TLS for encryption metadata for a given ciphersuite table
nx_secure_tls_packet_allocate	nx_secure_tls_packet_allocate(tls_session, pool_ptr, packet_ptr, wait_option); Allocates a packet for a TLS application such that it allows additional room for the TLS header

nx_secure_tls_remote_ certificate_allocate	nx_secure_tls_remote_certificate_allocate(tls_session, certificate, raw_certificate_buffer, buffer_size); Adds an uninitialized certificate instance to a TLS session for the purpose of allocating space for certificates provided by a remote host during a TLS session
nx_secure_tls_session_certificate_callback_set	nx_secure_tls_session_certificate_callback_set(tls_session, session); Sets up a function pointer that TLS will invoke when a certificate is received from a remote host, allowing the application to perform validation checks such as certificate revocation and certificate policy enforcement
nx_secure_tls_session_create	nx_secure_tls_session_create(session_ptr, cipher_table, metadata_area, metadata_size); Initializes a TLS session control block for later use in establishing a secure TLS session over a TCP socket or other lower-level networking protocol
nx_secure_tls_session_client_verify_disable	nx_secure_tls_session_client_verify_ disable(tls_session); Disables Client Certificate Verification for a particular TLS Session which previously had it enabled.
nx_secure_tls_session_client_verify_enable	nx_secure_tls_session_client_verify_enable(tls_session); Enables Client Certificate Verification for TLS Server instances. If enabled, the TLS Server will request and verify a remote TLS Client Certificate using all available crypto signature routines.
nx_secure_tls_session_delete	nx_secure_tls_session_delete(tls_session); Deletes a TLS session object, returning any resources to the system
nx_secure_tls_session_end	nx_secure_tls_session_end(tls_session, wait_option); Ends an active TLS session by sending the TLS CloseNotify alert to the remote host, then waiting for the response CloseNotify before returning.
nx_secure_tls_session_packet_buffer_set	nx_secure_tls_session_packet_buffer_set(session_ptr, buffer_ptr, buffer_size); Sets the buffer TLS uses to reassemble incoming messages which may span multiple TCP packets.

nx_secure_tls_session_protocol_oversion_override	nx_secure_tls_session_protocol_version_override(tls_session, protocol_version); Overrides the TLS protocol version to use for the TLS session. This allows for a different version of TLS to be utilized even if a newer version is enabled.
nx_secure_tls_session_receive	nx_secure_tls_session_receive(tls_session, packet_ptr_ptr, wait_option); Receives data from an active TLS session, handling all decryption and verification before returning the data to the caller in the supplied NX_PACKET structure
nx_secure_tls_session_reset	nx_secure_tls_session_reset(session_ptr); Resets a TLS session object, clearing out all data for initialization or re-use.
nx_secure_tls_session_send	nx_secure_tls_session_send(tls_session, packet_ptr, wait_option); Sends data using an active TLS session, handling all encryption and hashing before sending data over the established TCP socket connection
nx_secure_tls_session_start	nx_secure_tls_session_start(tls_session, tcp_socket, wait_option); Starts a TLS session given a TCP socket. The TCP connection must be established before calling this function or the TLS handshake will fail.
nx_secure_tls_session_time_function_set	nx_secure_tls_session_time_function_set(tls_session, time_func_ptr); Sets up a function pointer that TLS will invoke when it needs to get the current time, which is used in various TLS handshake messages and for verification of certificates.
nx_secure_tls_trusted_ certificate_add	nx_secure_tls_trusted_certificate_add(tls_session, certificate); Adds an initialized certificate to a TLS session for use as a trusted Root Certificate
nx_secure_tls_trusted_certificate_remove	nx_secure_tls_trusted_certificate_remove(tls_session, common_name, common_name_length); Removes a certificate instance from the trusted certificates store, keyed on the Common Name field.
nx_secure_tls_remote_certificate_free_all	nx_secure_tls_remote_certificate_free_all(NX_SECURE_TLS_SESSION *session_ptr); Release certificates previously registered with the TLS session.

**nx_secure_tls_psk_add	nx_secure_tls_psk_add(tls_session, pre_shared_key, psk_length, psk_identity, identity_length, hint, hint_length); Adds a pre-shared key (PSK) to a TLS session for use with a PSK ciphersuite. The second parameter is the PSK identity used during the TLS handshake to select the proper key.
**nx_secure_tls_psk_find	nx_secure_tls_psk_find(tls_session, psk_data, psk_length,psk_identity, identity_length); Finds a pre-shared key (PSK) in a TLS session for use with a PSK ciphersuite. The PSK is found using an "identity hint" that should match a field in the PSK structure in the TLS session.
**nx_secure_tls_client_psk_set	nx_secure_tls_client_psk_set(tls_session, pre_shared_key, psk_length, psk_identity, identity_length, hint, hint_length); Sets the pre-shared key (PSK) for a TLS Client in a TLS session control block for use with a remote server that is sing a PSK ciphersuite.
nx_secure_x509_certificate_initialize	nx_secure_x509_certificate_initialize(NX_SECURE_X509_CERT *certificate_ptr, const UCHAR *certificate_data, USHORT certificate_data_length, UCHAR *raw_data_buffer, USHORT buffer_size, const UCHAR *private_key_data, USHORT private_key_data_length, UINT private_key_type); Initialize X.509 Certificate for NetX Secure TLS
nx_secure_x509_common_name_dns_check	nx_secure_x509_common_name_dns_check(&certificate, dns_tld, dns_tld_length) Check DNS name against X.509 Certificate
nx_secure_x509_crl_revocation_ check	nx_secure_x509_crl_revocation_check(&crl_data, crl_length, &cert_store, &certificate) Check X.509 Certificate against a supplied Certificate Revocation List

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

** Requires that the property PSK Cipher Suite of the TLS Common component be enabled.

NetX Duo TLS Session Status Return Values

Please refer *NetX Secure TLS User Guide* section of *Azure RTOS NetX Duo Documentation* for details on status return values by NetX Secure TLS services and NetX secure X.509 Certificate error codes.

4.3.35.3 NetX Duo TLS Session Module Operational Overview

TLS uses TCP and provides secure communications for application layer protocols such as HTTP and MQTT. TLS can also be used in 'bare' TCP socket applications to send and receive TCP packets in a secure session to another TCP peer. The module guide for this project uses this simplified application of TLS to demonstrate a TLS Client TCP and TLS Server TCP sockets exchanging data to a TCP peer.

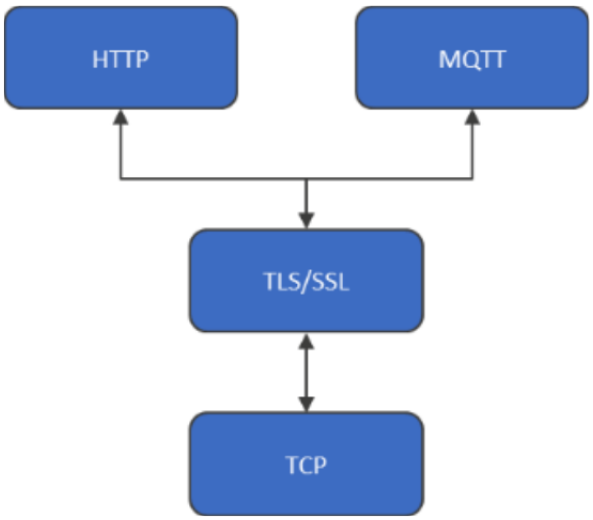


Figure 513: NetX Duo TLS Session Module TLS/SSL Layering

TLS does not have a well-known port number; instead, it uses the designated port number of the secure variant of the higher layer protocol. For example, port number 443 for secure HTTP, port number 8883 for MQTT, etc.

When a secure connection is established using TLS/SSL, for example using HTTPS, messages are exchanged between the client (which always initiates the connection) and a server. The first set of messages execute a Handshake Protocol after which the client and server can securely send/receive data bi-directionally, as shown in the following figure:

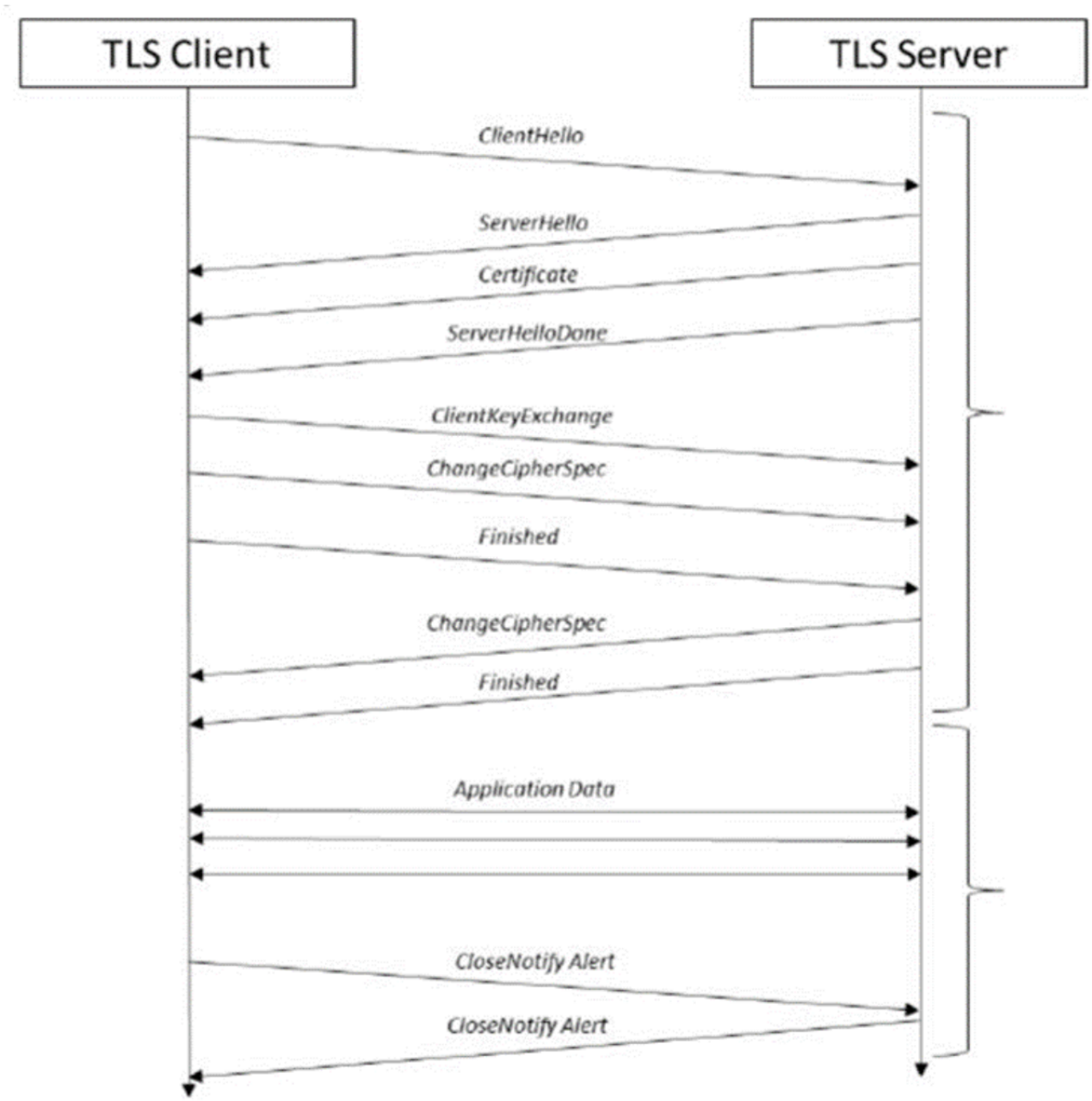


Figure 514: NetX Duo TLS Session Module TLS Protocol Sequences

NetX Duo TLS Session Module Important Operational Notes and Limitations

NetX Duo TLS Session Module Operational Notes

- Before the TLS session is created, the metadata buffer must be allocated. The size of meta data can be set using the properties pane. Also, the user can use NetX Duo Secure nx_secure_metadata_size_calculate API to calculate the required size of metadata buffer. The Metadata size is specified in the "Meta data size" of the TLS session component, or in the nx_secure_tls_session_create call; the default value is 4k but to handle most servers, 8k is recommended if the memory space is available.
- To associate a packet reassembly buffer to a TLS session, use the nx_secure_tls_session_packet_buffer_set API. The reassembly buffer is used to place the incoming TLS records which may span multiple TCP packets. If an incoming TLS record is

larger than the supplied buffer, the TLS session will end with an error. A reasonable packet buffer size is 6-8k.
- Also, before starting a TLS client session, the application must allocate memory for processing Server certificate data in the nx_secure_tls_remote_certificate_allocate call. A reasonable size for most certificates is 2k. The TLS client application should allow for 2-3 certificates from most servers.
- For any incoming certificate, the NetX Duo Secure TLS will perform basic X.509 path validation.
- Additionally, at each stage in the verification process the expiration date of each certificate is checked against the time provided by the application timestamp function. The nx_secure_tls_session_time_function_set API is used to optionally set up a function pointer for application timestamp function that TLS will invoke when it needs to get the current time. The current time is used in some TLS handshake messages for verification of certificates. If a timestamp function is registered with the TLS session, a timestamp will be used in the generation of the Server or Client Hello, and in verifying the remote certificate.
- Before attempting to reconnect to the same or another TLS server, the TLS client must clear the TLS session. This is most easily done by calling nx_secure_tls_session_end. It is recommended to delete the TLS session and recreate it before making another connection attempt. For applications using SSP 1.3.x, the nx_secure_tls_session_create call should be preceded by a memset call on the TLS session to clear the session completely: memset(tls_session_ptr, 0, sizeof(NXD_SECURE_TLS_SESSION));
- The user can use "Netx Duo TLS Common" module to configure Maximum RSA modulus size in bits. When user chooses Maximum RSA Modulus Size as 4096 bits then cryptographic operation for RSA such as encryption/decryption are done in software. Whereas if the user chooses Maximum RSA Modulus size as either 1024 or 2048 bits then cryptographic operations are done using Synergy Crypto Engine (SCE) hardware accelerator.
- TLS1.3 implementation in SSP currently supports only ECC based signature and ECDH key sharing.

NetX Duo TLS Session Module Limitations

- Due to the nature of embedded devices, some systems may not have adequate memory to support the maximum TLS record size of 16 KB. NetX Duo TLS Secure can handle 16KB records on devices with sufficient resources.
- NetX Duo Secure performs basic certificate verification only. NetX Duo TLS Secure will perform basic X.509 chain verification on a certificate to assure that the certificate is valid and signed by a trusted Certificate Authority, and can provide the certificate Common Name for the application to compare against the Top-Level Domain Name of the remote host. However, verification of certificate extensions and other data is the responsibility of the application implementer. Refer to the *Azure RTOS NetX Duo TLS Secure User Guide* available from the Synergy Gallery for more details.
- Software-based cryptography is processor-intensive and not available. Therefore, hardware-based cryptography is used for optimal performance of NetX Duo TLS Secure.
- Refer to the most recent *SSP release notes* for additional limitations on this module.

4.3.35.4 Including the NetX Duo TLS Session Module in an Application

This section describes how to include either or both the NetX Duo TLS Session module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo TLS Session module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

Note

Both TLS and DTLS session modules are included through the same stack as mentioned below.

User need not enable 'DTLS' property of 'NetX Duo TLS Common' to use TLS services.

Note

Both TLS and DTLS session modules are included through the same stack as mentioned below.

User has to enable 'DTLS' property of 'NetX Duo TLS Common' stack to use DTLS services.

NetX Duo TLS Session Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_tls_session0NetX Duo TLS/DTLS Session	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo TLS/DTLS Session

When the NetX Duo TLS/DTLS Session module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

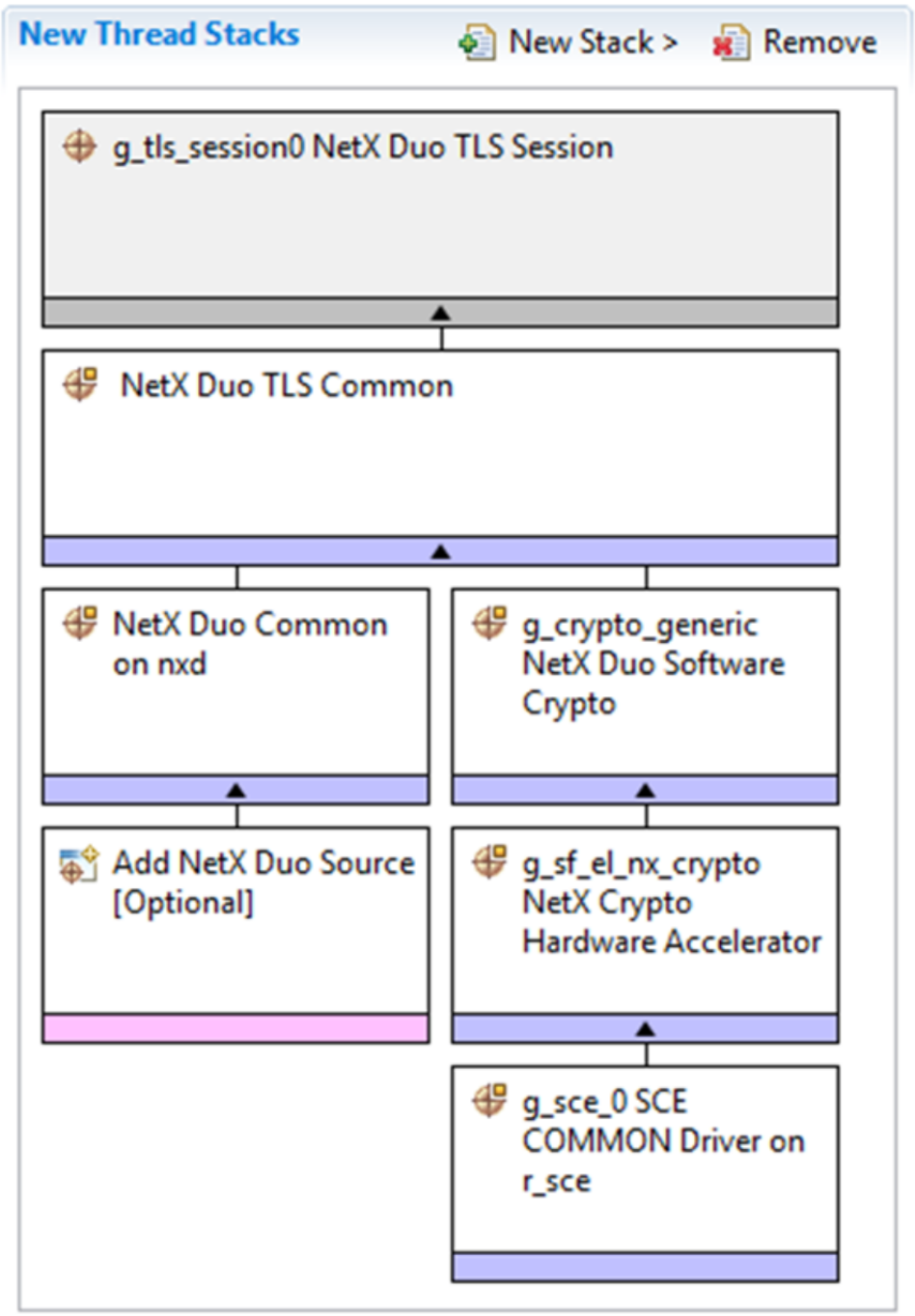


Figure 515: NetX Duo TLS Session Module Stack

4.3.35.5 Configuring the NetX Duo TLS Session Module

The NetX Duo TLS Session module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-

accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo TLS Session Module

ISDE Property	Value	Description
Security Protocol Name	TLS, DTLS Default : TLS	Security protocol Selection
Session Name	g_tls_session0	Module name
Meta data size	4000	Meta data size selection
Auto initialization	Enable, Disable Default : Enable	Auto initialization selection
*Name of Timestamp function	tls_timestamp_callback0	Name of timestamp function
**Name of Certificate Verification function	certificate_verification_callback0	Name of certificate verification function
Name of generated initialization function	tls_dtls_session_init0	Name of generated initialization function

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

*The Time Stamp function is discussed in section 3.1; it is omitted in this module guide in the interests of simplicity but it is typically used to check the expiration date of each certificate against the time provided by the application. Not having a timestamp callback may cause interoperability problems and reduce the security of production release TLS application sessions.

**The Certificate Verification function provides the certificate being verified to the application so additional verification steps may be performed. It is set to NULL in the module guide project for the sake of simplicity but in a real TLS session it would be verification tool.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different addresses for the Ethernet port. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note

Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo TLS Session Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent

user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo TLS Common

ISDE Property	Value	Description
Crypto Engine	Hardware	Crypto engine selection
Self Signed Certificates	Enable, Disable Default: Disable	Self signed certificates selection
PSK Cipher Suite	Enable, Disable Default: Disable	PSK cipher suite selection
ECC Cipher Suite	Enable, Disable Default: Disable	ECC cipher suite selection
AES-GCM Data Buffer Size(Bytes)	Default: 2048	Maximum size of internal data buffer for encryption/decryption process in AES-GCM based cipher-suites.
X509 Strict Name Compare	Enable, Disable Default: Disable	X509 strict name compare selection
X509 Extended Distinguished Names	Enable, Disable Default: Disable	X509 extended distinguished names selection
X509 Certificate Revocation List Check	Enable, Disable Default: Enable	Revocation list check selection
X509	Enable, Disable Default: Enable	X509 feature selection

AEAD Cipher	Enable, Disable Default: Disable	AEAD cipher support selection
AEAD Cipher Check	Enable, Disable Default: Disable	AEAD cipher check selection for the AEAD cipher suites other than AES-CCM or AES-GCM
Additional AEAD Cipher ID	Default: NX_FALSE	Name of a valid additional AEAD cipher ID selection
SCSV Cipher suite	Enable, Disable Default: Disable	SCSV cipher suite selection
Maximum RSA Modulus size (bits)	1024, 2048, 3072, 4096 Default: 4096	Maximum RSA modulus size (bits) selection
TLS v 1.3	Enable, Disable Default: Disable	TLS v 1.3 selection
TLS Version Downgrade	Enable, Disable Default: Enable	TLS version downgrade selection
DTLS	Enable, Disable Default: Disable	DTLS enabling selection
DTLS Cookie Length	32	DTLS cookie length selection
DTLS Retransmit Retries	10	Number of DTLS retransmit retries
DTLS Initial Retransmit Rate(Sec)	1	DTLS initial retransmit rate in seconds
DTLS Maximum Retransmit Rate(Sec)	60	DTLS maximum retransmit rate in seconds
Maximum PSK ID Size	Default: 20	Maximum PSK ID size selection (bytes)
Maximum PSK Keys	Default: 5	Maximum PSK Keys selection
Maximum Size of PSK	Default: 20	Maximum Size of PSK Selection (bytes)
Minimum TLS X509 Certificate Size	Default: 256	Minimum TLS X509 Certificate size selection (bytes)
Minimum TLS Message Buffer Size	Default: 4000	Minimum TLS Message Buffer Size selection (bytes)
Size of TLS Pre-Master Secret	Default: 48	Size of TLS Pre-Master Secret selection (bytes)

Secure Key Clear	Enable, Disable Default: Disable	Secure key clear selection
TLS SNI Extension	Enable, Disable Default: Enable	Server name indication selection
Server Mode	Enable, Disable Default: Enable	Server mode selection
Client Mode	Enable, Disable Default: Enable	Client mode selection
Client Certificate Verify	Enable, Disable Default: Disable	Client certificate verification selection
Name of generated initialization function	nx_secure_common_init	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
TLS 1.2 will be enabled by default in any NetX Duo TLS applications.
The AES-GCM data Buffer size refers to the internal input/output buffer used for the data encryption/decryption process. The value provided should be greater than or equal to the application data packet size, considering the available RAM space.
The Pre-Master Secret size should be at least 66 bytes when ECC cipher suites are used.
When the AEAD cipher check is enabled, we recommend that the user provide a valid algorithm ID in the additional AEAD cipher ID field

Configuration Settings for the NetX Duo Software Crypto

ISDE Property	Value	Description
Name	g_crypto_generic	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Crypto Hardware Accelerator

ISDE Property	Value	Description
Name	g_sf_el_nx_crypto	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have

different default values and available configuration settings.

Configuration Settings for the SCE Common Driver

ISDE Property	Value	Description
Name	g_sce_0	Module name
Endian Flag	CRYPTO_WORD_ENDIAN_BIG, CRYPTO_WORD_ENDIAN_LITTLE Default: CRYPT_WORD_ENDIAN_BIG	Endian flag selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo TLS Session Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo TLS Session Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection

REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.35.6 Using the NetX Duo TLS Session Module in an Application

Autogenerated code includes the initialization function with the name as specified in the *Name of generated initialization function property*, for example, tls_dlts_session_init0(). The TLS Session instance variable specified using the *Name* property of the TLS Session is passed as input to the initialization function and this instance variable can be passed as input to several APIs of the TLS layer. The initialization function internally calls the nx_secure_tls_session_create API to create a TLS session. Calls to the initialization function will be enabled or disabled depending on the *Auto Initialization* property value.

The steps in using the NetX Duo TLS Session module in a typical application are:

1. Packet the reassembly buffer for TLS session using the nx_secure_tls_session_packet_buffer_set API
2. Initialize the root CA certificate using the nx_secure_x509_certificate_initialize API
3. Load the root CA certificate to a trusted store using the nx_secure_tls_trusted_certificate_add API
4. Allocate space for certificates sent by remote server using the nx_secure_tls_remote_certificate_allocate API
5. Start the TLS session using the nx_secure_tls_session_start API
6. Send data over the secure connection using the nx_secure_tls_session_send API
7. Receive data over the secure connection using the nx_secure_tls_session_receive API
8. End the session using the nx_secure_tls_session_end API

The following figure illustrates common steps in a typical operational flow diagram:

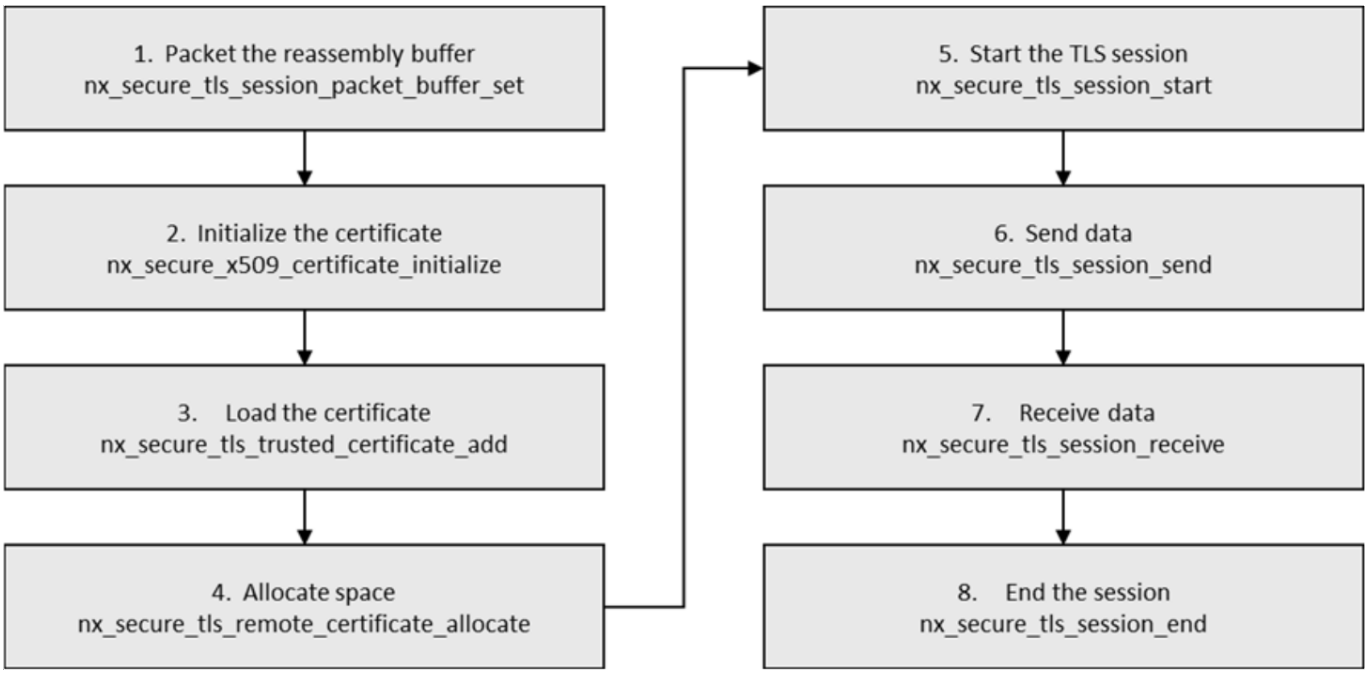


Figure 516: Flow Diagram of a Typical NetX Duo TLS Session Module Application

4.3.36 NetX Duo DTLS Session

4.3.36.1 NetX Duo DTLS Session Introduction

DTLS is closely coupled with TLS, as the underlying security mechanisms are shared between the protocols. However,TLS is designed to work over a transport layer protocol that guarantees about packet delivery and order (almost always TCP in practice) and will not function over an unreliable protocol like UDP. It is precisely because of UDP that DTLS was introduced. DTLS was designed to handle the unreliable nature of UDP and similar protocols. It does this by including ordering and reliability logic (e.g. retransmission of dropped data) similar to reliable protocols like TCP

The NetX Duo DTLS Session Module provides a high-level API for Datagram Transport Layer Security (DTLS) protocol-based clients and servers. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo DTLS Session module is based on ThreadX "NetX Duo Secure," which implements the Datagram Transport Layer Security (DTLS) protocol as described in RFCs 6347 (version 1.2). NetX Duo Secure also includes routines for basic X.509 (RFC 5280).

The DTLS protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

- Encryption: The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanism such as AES (Advanced Encryption Standard is used for data encryption.
- Authentication: A mechanism to check the peer's identity using certificates
- Integrity: A mechanism to detect message tampering and forgery to ensure that

connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity.

NetX Duo DTLS Session Module Features

- RFC 6347 The Datagram Transport Layer Security (DTLS) Protocol version 1.2
- RFC 5280 X.509 PKI Certificates (v3)
- RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Datagram Transport Layer Security (DTLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for DTLS
- Supports DTLS extensions for the following:
    1. Secure Renegotiation Indication: This extension mitigates a Man-in-the-Middle attack vulnerability that could occur during a renegotiation handshake.
    2. Server Name Indication: This extension allows a DTLS Client to supply a specific DNS name to a DTLS Server, allowing the server to select the correct credentials (assumes the server has multiple identity certificates and network entry points).
    3. Signature Algorithms: This extension enables a DTLS Client to provide a list of acceptable signature and hash algorithms to a DTLS Server
- Supports X.509 extensions for the following:
    1. Key Usage: Provides acceptable uses for a certificate's public key in a bitfield
    2. Extended Key Usage: Provides additional acceptable uses for a certificate's public key using OIDs
- Certificate verification: Supports SHA-1, SHA-256, SHA-384 and SHA-512 signature hashing algorithms
- Subject Alternative Name: Provides alternative DNS names that are also represented by the certificate.

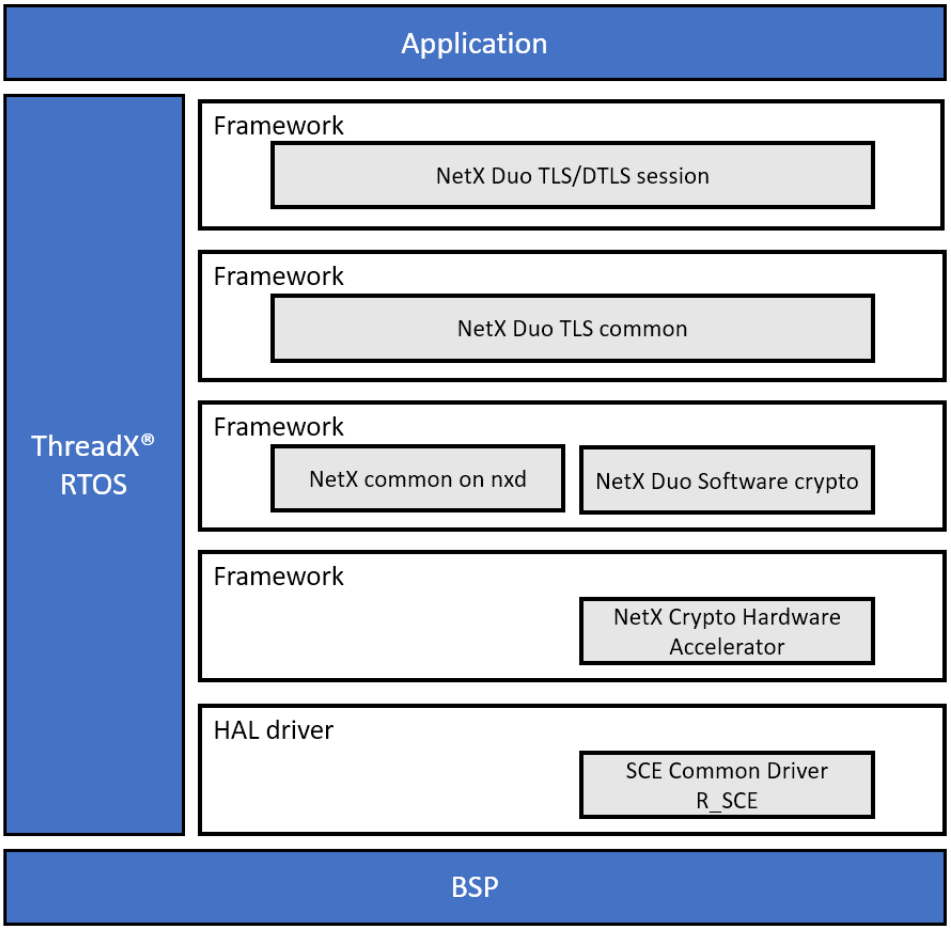


Figure 517: NetX Duo Secure DTLS Session Module Block Diagram

4.3.36.2 NetX Duo DTLS Session Module APIs Overview

The NetX Duo DTLS Session module defines APIs for creating and setting up a DTLS security session. A complete list of the available APIs, an example API call and a short description of each can be found in the following table.

Function Name	Example API Call and Description
nx_secure_dtls_client_session_start	nx_secure_dtls_client_session_start(NX_SECURE_DTLS_SESSION *dtls_session, NX_UDP_SOCKET *udp_socket, NXD_ADDRESS *ip_address, UINT port, UINT wait_option); Starts DTLS client session connecting to the provided server address and port using UDP socket.
nx_secure_dtls_packet_allocate	nx_secure_dtls_packet_allocate(NX_SECURE_DTLS_SESSION *session_ptr, NX_PACKET_POOL *pool_ptr, NX_PACKET **packet_ptr, ULONG wait_option); Allocate a packet for a NetX Secure DTLS Session

nx_secure_dtls_psk_add	nx_secure_dtls_psk_add(NX_SECURE_DTLS_SESSION *session_ptr, UCHAR *pre_shared_key, UINT psk_length, UCHAR *psk_identity, UINT identity_length, UCHAR *hint, UINT hint_length); Add a Pre-Shared Key to a NetX Secure DTLS Session
nx_secure_dtls_server_create	nx_secure_dtls_server_create(NX_SECURE_DTLS_SERVER *server_ptr, NX_IP *ip_ptr, UINT port, ULONG timeout, VOID *session_buffer, UINT session_buffer_size, const NX_SECURE_TLS_CRYPTO *crypto_table, VOID *crypto_metadata_buffer, ULONG crypto_metadata_size, UCHAR *packet_reassembly_buffer, UINT packet_reassembly_buffer_size, UINT (*connect_notify)(NX_SECURE_DTLS_SESSION *dtls_session, NXD_ADDRESS *ip_address, UINT port), UINT (*receive_notify)(NX_SECURE_DTLS_SESSION *dtls_session))); Create a NetX Secure DTLS Server
nx_secure_dtls_server_delete	nx_secure_dtls_server_delete(NX_SECURE_DTLS_SERVER *server_ptr); Free up resources used by a NetX Secure DTLS Server
nx_secure_dtls_server_local_certificate_add	nx_secure_dtls_server_local_certificate_add(NX_SECURE_DTLS_SERVER *server_ptr, NX_SECURE_X509_CERT *certificate, UINT cert_id); Add a local server identity certificate to a NetX Secure DTLS Server
nx_secure_dtls_server_local_certificate_remove	nx_secure_dtls_server_local_certificate_remove(NX_SECURE_DTLS_SERVER *server_ptr, UCHAR *common_name, UINT common_name_length, UINT cert_id); Remove a local server identity certificate from a NetX Secure DTLS Server
nx_secure_dtls_server_notify_set	nx_secure_dtls_server_notify_set(NX_SECURE_DTLS_SERVER *server_ptr, UINT (*disconnect_notify)(NX_SECURE_DTLS_SESSION *dtls_session), UINT (*error_notify)(NX_SECURE_DTLS_SESSION *dtls_session, UINT error_code)); Assign optional notification callback routines to a NetX Secure DTLS Server
nx_secure_dtls_server_psk_add	nx_secure_dtls_server_psk_add(NX_SECURE_DTLS_SERVER *server_ptr, UCHAR *pre_shared_key, UINT psk_length, UCHAR *psk_identity, UINT identity_length, UCHAR *hint, UINT hint_length); Add a Pre-Shared Key to a NetX Secure DTLS Server

nx_secure_dtls_server_session_send	nx_secure_dtls_server_session_send(NX_SECURE_DTLS_SESSION *session_ptr, NX_PACKET *packet_ptr); Send data over a DTLS session established with a NetX Secure DTLS Server
nx_secure_dtls_server_session_start	nx_secure_dtls_server_session_start(NX_SECURE_DTLS_SESSION *session_ptr, UINT wait_option); Start a DTLS Session from a NetX Secure DTLS Server
nx_secure_dtls_server_start	nx_secure_dtls_server_start(NX_SECURE_DTLS_SERVER *server_ptr); Start a NetX Secure DTLS Server instance listening on the configured UDP port
nx_secure_dtls_server_stop	nx_secure_dtls_server_stop(NX_SECURE_DTLS_SERVER *server_ptr); Stop an active NetX Secure DTLS Server instance
nx_secure_dtls_server_trusted_certificate_add	nx_secure_dtls_server_trusted_certificate_add(NX_SECURE_DTLS_SERVER *server_ptr, NX_SECURE_X509_CERT *certificate, UINT cert_id); Add a trusted CA certificate to a NetX Secure DTLS Server
nx_secure_dtls_server_trusted_certificate_remove	nx_secure_dtls_server_trusted_certificate_remove(NX_SECURE_DTLS_SERVER *server_ptr, UCHAR *common_name, UINT common_name_length, UINT cert_id); Remove a trusted CA certificate from a NetX Secure DTLS Server
nx_secure_dtls_server_x509_client_verify_configure	nx_secure_dtls_server_x509_client_verify_configure (NX_SECURE_DTLS_SERVER *server_ptr, UINT certs_per_session, UCHAR *certs_buffer, ULONG buffer_size); Configure a NetX Secure DTLS Server to request and verify client certificates
nx_secure_dtls_server_x509_client_verify_disable	nx_secure_dtls_server_x509_client_verify_disable (NX_SECURE_DTLS_SERVER *server_ptr); Disables client X.509 certificate verification for a NetX Secure DTLS Server
nx_secure_dtls_session_client_info_get	nx_secure_dtls_session_client_info_get(NX_SECURE_DTLS_SESSION *session_ptr, NXD_ADDRESS *client_ip_address, UINT *client_port, UINT *local_port); Get remote client information from a DTLS Server session

nx_secure_dtls_session_create	nx_secure_dtls_session_create(NX_SECURE_DTLS_SESSION *dtls_session, const NX_SECURE_TLS_CRYPTO *crypto_table, VOID *metadata_buffer, ULONG metadata_size, UCHAR *packet_reassembly_buffer, UINT packet_reassembly_buffer_size, UINT certs_number, UCHAR *remote_certificate_buffer, ULONG remote_certificate_buffer_size)); Create and configure a NetX Secure DTLS Session
nx_secure_dtls_session_delete	nx_secure_dtls_session_delete(NX_SECURE_DTLS_SESSION *dtls_session); Free up resources used by a NetX Secure DTLS Session
nx_secure_dtls_session_end	nx_secure_dtls_session_end(NX_SECURE_DTLS_SESSION *dtls_session, UINT wait_option); Shut down an active NetX Secure DTLS Session
nx_secure_dtls_session_local_certificate_add	nx_secure_dtls_session_local_certificate_add(NX_SECURE_DTLS_SESSION *session_ptr, NX_SECURE_X509_CERT *certificate, UINT cert_id); Add a local identity certificate to a NetX Secure DTLS Session
nx_secure_dtls_session_local_certificate_remove	nx_secure_dtls_session_local_certificate_remove(NX_SECURE_DTLS_SESSION *session_ptr, UCHAR *common_name, UINT common_name_length, UINT cert_id); Remove a local identity certificate from a NetX Secure DTLS Session
nx_secure_dtls_session_receive	nx_secure_dtls_session_receive(NX_SECURE_DTLS_SESSION *dtls_session, NX_PACKET **packet_ptr_ptr, UINT wait_option); Receive application data over an established NetX Secure DTLS Session
nx_secure_dtls_session_reset	nx_secure_dtls_session_reset(NX_SECURE_DTLS_SESSION *dtls_session); Clear data in an NetX Secure DTLS Session instance
nx_secure_dtls_session_send	nx_secure_dtls_session_send(NX_SECURE_DTLS_SESSION *session_ptr, NX_PACKET *packet_ptr, NXD_ADDRESS *ip_address, UINT port); Send data over a DTLS session
nx_secure_dtls_session_trusted_certificate_add	nx_secure_dtls_session_trusted_certificate_add(NX_SECURE_DTLS_SESSION *session_ptr, NX_SECURE_X509_CERT *certificate, UINT cert_id); Add a trusted CA certificate to a NetX Secure DTLS Session

nx_secure_dtls_session_trusted_certificate_remove	nx_secure_dtls_session_trusted_certificate_remove(NX_SECURE_DTLS_SESSION *session_ptr, UCHAR *common_name, UINT common_name_length, UINT cert_id); Remove a trusted CA certificate from a NetX Secure DTLS Session

Note

For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS User's Manual in the References section.

NetX Duo DTLS Session Status Return Values

Please refer *NetX Secure DTLS User Guide* section of *Azure RTOS NetX Duo Documentation* for details on status return values by NetX Secure DTLS services and NetX secure X.509 Certificate error codes.

4.3.36.3 NetX Duo DTLS Session Module Operational Overview

DTLS can be used in 'bare' UDP socket applications to send and receive UDP packets in a secure session to another UDP peer.

When a secure connection is established using DTLS, messages are exchanged between the client (which always initiates the connection) and a server. The first set of messages execute a Handshake Protocol after which the client and server can securely send/receive data bi-directionally, as shown in the following figure:

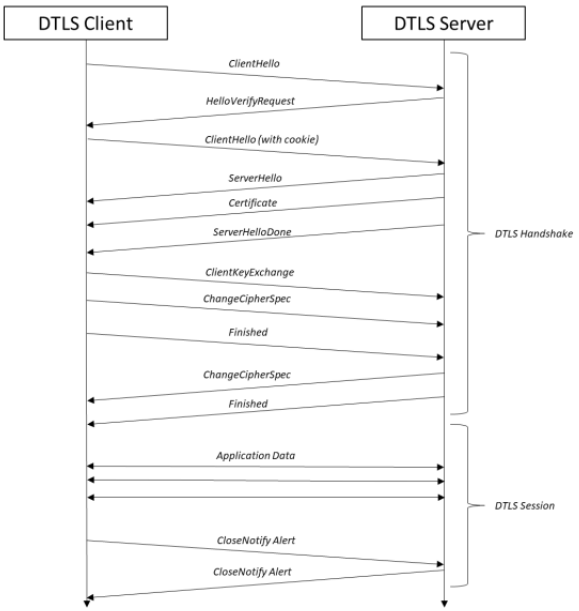


Figure 518: DTLS Protocol

NetX Duo DTLS Session Module Important Operational Notes and Limitations

NetX Duo DTLS Session Module Operational Notes

- For any incoming certificate, the NetX Duo Secure DTLS will perform basic X.509 path validation.

- Before attempting to reconnect to the same or another DTLS server, the DTLS client must clear the DTLS session. This is most easily done by calling nx_secure_dtls_session_end. It is recommended to delete the DTLS session and recreate it before making another connection attempt.
- The user can use "NetX Duo TLS Common" module to configure Maximum RSA modulus size in bits. When user chooses Maximum RSA Modulus Size as 4096 bits then cryptographic operation for RSA such as encryption/decryption are done in software. Whereas if the user chooses Maximum RSA Modulus size as either 1024 or 2048 bits then cryptographic operations are done using Synergy Crypto Engine (SCE) hardware accelerator.

NetX Duo DTLS Session Module Limitations

1. Due to the nature of embedded devices, some applications may not have the resources to support the maximum DTLS record size of 16KB. NetX Secure can handle 16KB records on devices with sufficient resources.
2. Minimal certificate verification. NetX Secure will perform basic X.509 chain verification on a certificate to assure that the certificate is valid and signed by a trusted Certificate Authority and can provide the certificate Common Name for the application to compare against the Top-Level Domain Name of the remote host. However, verification of certificate extensions and other data is the responsibility of the application implementer. Refer to the *Azure RTOS NetX Duo DTLS Secure User Guide* available from the Synergy Gallery for more details.
3. Software-based cryptography is processor-intensive and is currently not supported in SSP. Therefore, hardware-based cryptography is used for optimal performance of NetX Secure DTLS.
4. Auto initialization option for DTLS configurable properties is not supported and must be disabled.
5. Refer to the most recent SSP release notes for additional limitations on this module.

4.3.36.4 Including the NetX Duo DTLS Session Module in an Application

This section describes how to include the NetX Duo DTLS Session module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo DTLS Session module to an application, simply add it to a thread using the stacks selection sequence given in the following table and enable 'DTLS' property of 'NetX Duo TLS Common' stack.

Note

Both TLS and DTLS session modules are included through the same stack as mentioned below.

User has to enable 'DTLS' property of 'NetX Duo TLS Common' stack to use DTLS services.

Resource	ISDE Tab	Stacks Selection Sequence
g_tls_session0NetX Duo TLS/DTLS Session	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo TLS/DTLS Session

When the NetX Duo TLS/DTLS Session module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the new icon and displays possible choices.

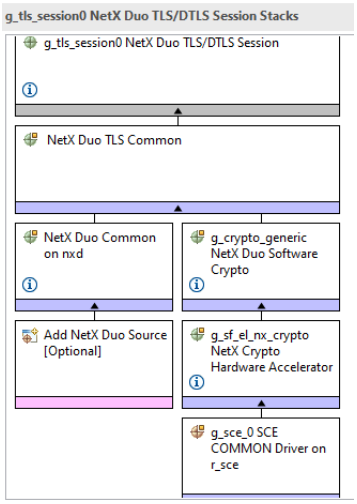


Figure 519: NetX Duo DTLS Session Module Stack

4.3.36.5 Configuring the NetX Duo DTLS Session Module

The NetX Duo DTLS Session module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo DTLS Session Module

ISDE Property	Value**	**Description
Security Protocol Name	TLS, DTLS Default: TLS	Security protocol selection
Session Name	g_tls_session0	Module name

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Meta data size	18000	Size of meta data
Name of Timestamp function	tls_timestamp_callback0	Name of timestamp function
Name of Certificate Verification function	certificate_verification_callback0	Name of certificate verification function
Name of generated initialization function	tls_dtls_session_init0	Name of generated initialization function

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Auto Initialization has to be disabled for DTLS Session.
The properties 'Meta data size', 'Name of Timestamp function', 'Name of Certificate Verification function', 'Name of generated initialization function' are applicable only for TLS Session.

Configuration Settings for the NetX Duo DTLS Session Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo TLS Common

ISDE Property	Value	Description
Crypto Engine	Hardware	Crypto engine selection
Self Signed Certificates	Enable, Disable Default: Disable	Self signed certificates selection
PSK Cipher Suite	Enable, Disable Default: Disable	PSK cipher suite selection

ECC Cipher Suite	Enable, Disable Default: Disable	ECC cipher suite selection
AES-GCM Data Buffer Size(Bytes)	Default: 2048	Maximum size of internal data buffer for encryption/decryption process in AES-GCM based cipher-suites.
X509 Strict Name Compare	Enable, Disable Default: Disable	X509 strict name compare selection
X509 Extended Distinguished Names	Enable, Disable Default: Disable	X509 extended distinguished names selection
X509 Certificate Revocation List Check	Enable, Disable Default: Enable	Revocation list check selection
X509	Enable, Disable Default: Enable	X509 feature selection
AEAD Cipher	Enable, Disable Default: Disable	AEAD cipher support selection
AEAD Cipher Check	Enable, Disable Default: Disable	AEAD cipher check selection for the AEAD cipher suites other than AES-CCM or AES-GCM
SCSV Cipher suite	Enable, Disable Default: Disable	SCSV cipher suite selection
Maximum RSA Modulus size (bits)	1024, 2048, 3072, 4096 Default: 4096	Maximum RSA modulus size (bits) selection
TLS v 1.3	Enable, Disable Default: Disable	TLS v 1.3 selection
TLS Version Downgrade	Enable, Disable Default: Enable	TLS version downgrade selection
DTLS	Enable, Disable Default: Disable	DTLS enabling selection
DTLS Cookie Length	32	DTLS cookie length selection
DTLS Retransmit Retries	10	Number of DTLS retransmit retries
DTLS Initial Retransmit Rate(Sec)	1	DTLS initial retransmit rate in seconds

DTLS Maximum Retransmit Rate(Sec)	60	DTLS maximum retransmit rate in seconds
Maximum PSK ID Size	Default: 20	Maximum PSK ID size selection (bytes)
Maximum PSK Keys	Default: 5	Maximum PSK Keys selection
Maximum Size of PSK	Default: 20	Maximum Size of PSK Selection (bytes)
Minimum TLS X509 Certificate Size	Default: 256	Minimum TLS X509 Certificate size selection (bytes)
Minimum TLS Message Buffer Size	Default: 4000	Minimum TLS Message Buffer Size selection (bytes)
Size of TLS Pre-Master Secret	Default: 48	Size of TLS Pre-Master Secret selection (bytes)
Secure Key Clear	Enable, Disable Default: Disable	Secure key clear selection
TLS SNI Extension	Enable, Disable Default: Enable	Server name indication selection
Server Mode	Enable, Disable Default: Enable	Server mode selection
Client Mode	Enable, Disable Default: Enable	Client mode selection
Client Certificate Verify	Enable, Disable Default: Disable	Client certificate verification selection
Name of generated initialization function	nx_secure_common_init	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
DTLS should be enabled to use DTLS V 1.2 protocol.
The AES-GCM data Buffer size refers to the internal input/output buffer used for the data encryption/decryption process. The value provided should be greater than or equal to the application data packet size, considering the available RAM space.
The Pre-Master Secret size should be at least 66 bytes when ECC cipher suites are used.

Configuration Settings for the NetX Duo Software Crypto

ISDE Property	Value	Description
Name	g_crypto_generic	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Crypto Hardware Accelerator

ISDE Property	Value	Description
Name	g_sf_el_nx_crypto	Module name

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SCE Common Driver

ISDE Property	Value	Description
Name	g_sce_0	Module name
Endian Flag	CRYPTO_WORD_ENDIAN_BIG, CRYPTO_WORD_ENDIAN_LITTLE Default: CRYPT_WORD_ENDIAN_BIG	Endian flag selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo DTLS Session Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo DTLS Session Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.36.6 Using the NetX Duo DTLS Session Module in an Application

Note

User cannot enable Auto Initialization for DTLS session from SSP configurator. Hence the client and server initialization functions has to be invoked from the application.

The steps in using the NetX Duo DTLS Session module in a typical client application are:

1. Create a DTLS session using the nx_secure_dtls_session_create API.
2. Initialize the root CA certificate using the nx_secure_x509_certificate_initialize API
3. Load the root CA certificate to a trusted store using the nx_secure_dtls_session_trusted_certificate_add API
4. Start DTLS client session using nx_secure_dtls_client_session_start API.

5. Allocate a DTLS packet to send some encrypted data to the server using nx_secure_dtls_packet_allocate API.
6. Populate the packet with some data using nx_packet_data_appendAPI.
7. Send data over the secure connection to the server using nx_secure_dtls_session_sendAPI.
8. Receive data over the secure connection using nx_secure_dtls_session_receive API from the server.
9. End the session using nx_secure_dtls_session_endAPI.

The following figure illustrates common steps of client application in a typical operational flow diagram:

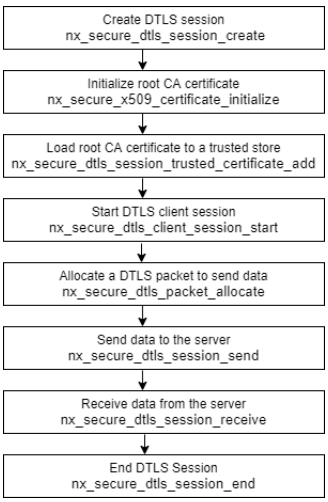


Figure 520: Flow Diagram of a Typical NetX Secure DTLS Session Module Client Application

The steps in using the NetX Duo DTLS Session module in a typical server application are:

1. Create a DTLS server instance using nx_secure_dtls_server_create API. User has to define Connect and Receive Notify callback functions which will be passed as parameters to nx_secure_dtls_server_create API.
2. Initialize local server identity certificate with key and add to server using nx_secure_x509_certificate_initialize API.
3. Add local server identity certificate to DTLS server using nx_secure_dtls_server_local_certificate_addAPI. API.
4. Start DTLS server using nx_secure_dtls_server_startAPI.
5. If there is a connection attempt from the client, start DTLS server session using nx_secure_dtls_server_session_startAPI.
6. Receive the data from connected client using nx_secure_dtls_session_receiveAPI.
7. Send the response to client using nx_secure_dtls_server_session_sendAPI
8. Once server processing is done, stop server instance from accepting connection requests using nx_secure_dtls_server_stopAPI.
9. Delete DTLS server session using nx_secure_dtls_server_deleteAPI.

The following figure illustrates common steps of server application in a typical operational flow diagram:

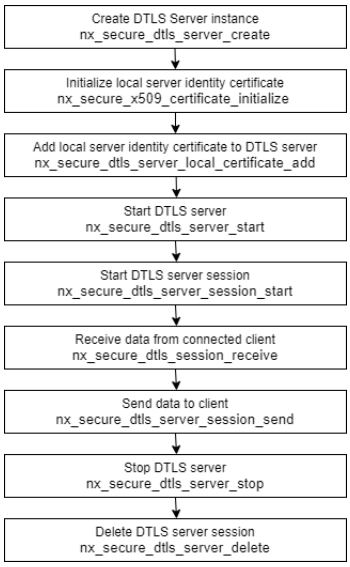


Figure 521: Flow Diagram of a Typical NetX Secure DTLS Session Module Server Application

4.3.37 NetX Duo mDNS/DNS-SD

4.3.37.1 NetX Duo mDNS/DNS-SD Introduction

Multicast DNS/DNS-SD (mDNS) is a protocol that helps in resolving the hostname to IP address in a local network. The mDNS and DNS-SD are protocols designed to augment the traditional DNS service. mDNS provides hostname and service lookup to the nodes on the local network. Each node uses an IPv4 or IPv6 multicast channel to announce services it offers to its neighbors, responds to queries from its neighbors, and sends queries on behalf of its applications. Throughout the document, the term mDNS refers to the services that cover both the mDNS specification and the DNS-SD specification.

NetX Duo mDNS/DNS-SD Module Features

- NetX Duo mDNS/DNS-SD is compliant with RFC 6762 and RFC 6763.
- NetX Duo mDNS/DNS-SD supports both IPv4 and IPv6 networks.
- Optional creation of separate packet pool for mDNS operations.
- Provides high-level APIs for:
    - Creating and deleting mDNS instances.
    - Creating and deleting local services.
    - Perform single shot and continuous service discovery.
    - Service lookup from the local peer service cache.
    - Setting cache and services notify callbacks.

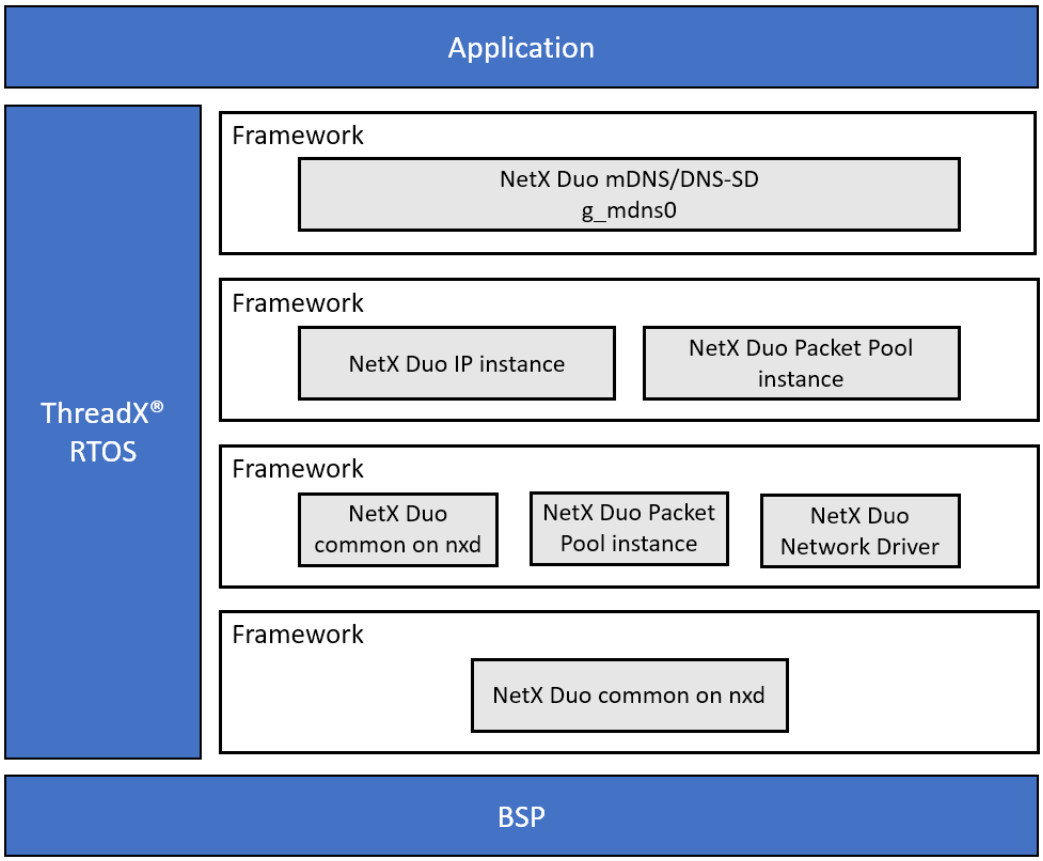


Figure 522: NetX Duo mDNS/DNS-SD Module Block Diagram

Note

In the figure above, the NetX Duo Network Driver modules has multiple implementation options available. See the description just after the module stack figure in Including the NetX Duo mDNS/DNS-SD Module in an Application for additional details.

4.3.37.2 NetX Duo mDNS/DNS-SD Module APIs Overview

The NetX Duo mDNS/DNS-SD Support module defines APIs for creating the mDNS instances, setting domain name, adding mDNS services, performing a one-shot and continuous discovery. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

NetX Duo mDNS Module API Summary

Function Name	Example API Call and Description
nx_mdns_create	nx_mdns_create(&my_mdns, &my_ip, &my_packet_pool, priority, stack_ptr, sizeof(stack_ptr), "NETX-MDNS-HOST", local_cache_ptr, sizeof(local_cache_ptr), peer_cache_ptr, sizeof(peer_cache_ptr), probing_notify); Creates an mDNS instance on the specific IP instance and associated resources.

nx_mdns_delete	nx_mdns_delete(&my_mdns); deletes the mDNS instance and frees its resources.
nx_mdns_enable	nx_mdns_enable(&my_mdns, interface_index); Enables mDNS service on a specific physical interface. Once the service is enabled, the mDNS module first probes all its unique service names on the network before responding to queries received on the interface.
nx_mdns_disable	nx_mdns_disable(&my_mdns, interface_index); Disables mDNS service on the specific physical interface. Once the service is disabled, the mDNS sends "goodbye" messages for every local service to the network that is attached to the interface, so the neighboring nodes are notified.
nx_mdns_cache_notify_set	nx_mdns_cache_notify_set(&my_mdns, cache_full_nofiy_cb); This service installs a user-supplied callback function, which is invoked when either the local service cache or peer service cache becomes full.
nx_mdns_cache_notify_clear	nx_mdns_cache_notify_clear(&my_mdns); Clears a user-supplied service cache notify callback function.
nx_mdns_domain_name_set	nx_mdns_domain_name_set(&my_mdns, "home"); This service sets up the default local domain name. When the mDNS instance is created, the default local domain name is set to ".local". This API allows an application to overwrite the default local domain name.
nx_mdns_service_announcement_timing_set	nx_mdns_service_announcement_timing_set(&my_mdns, t, p, k, retrans_interval, period_interval, max_time); This service reconfigures the timing parameters employed by mDNS when sending the service announcements. Publish period starts from t ticks and can be expanded telescopically with 2 to the power of k factor. The number of repetitions per advertisement is p, the interval between each repeated advertisement is *interval* ticks, and Maximum announcement period is max_time.

nx_mdns_service_add	nx_mdns_service_add(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL, NX_NULL, 0, priority, weight, port,is_unique, interface); This API registers a service offered by the application. If the flag is_unique is set, mDNS probes the service name to make sure it is unique on the local network before starting to announce the service on the network.
nx_mdns_service_delete	nx_mdns_service_delete(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL); This API deletes a previous registered service. As the service is deleted, "goodbye" messages are sent to the local network so the neighboring nodes are notified.
nx_mdns_service_one_shot_query	nx_mdns_service_one_shot_query(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL, service_ptr, wait_option); This service performs a one-shot mDNS query. If the specified service is found in the peer service cache, the first instance is returned. If no services are found in the local peer service cache, the mDNS module issues a query command and waits for a response.
nx_mdns_service_continuous_query	nx_mdns_service_continuous_query(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL); This service starts a continuous query. Note that the service returns immediately. After issuing a continuous query, the application may retrieve service records by using the API nx_mdns_service_lookup.
nx_mdns_service_query_stop	nx_mdns_service_query_stop(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL); Terminates the previous issued continuous service discovery.
nx_mdns_service_lookup	nx_mdns_service_lookup(&my_mdns, "NETX-SERVICE", "_http._tcp", NX_NULL, 0, service_ptr); This service looks up services matching the instance name (if provided) and the type of service in the local peer service cache.
nx_mdns_service_ignore_set	nx_mdns_service_ignore_set(&my_mdns, service_mask); This API configures a mask to ignore services specified by the service_mask bitmask. User may optionally use the service_mask to select service types it does not wish to be cached.

nx_mdns_service_notify_set	nx_mdns_service_notify_set(&my_mdns, service_mask, service_change_notify); This API configures a service change notify callback function. This callback function is invoked when a service offered by other nodes on the network is added, changed, or is no longer available.
nx_mdns_service_notify_clear	nx_mdns_service_notify_clear(&my_mdns); Clear the service change notify callback function.
nx_mdns_host_address_get	nx_mdns_host_address_get(&my_mdns, "MDNS-Host", &ipv4_address, ipv6_address, 500); This service performs an mDNS query on host IPv4 and IPv6 addresses. If the address of the specified hostname is found in the peer service cache, the address is returned. If no address is found in the peer service cache, the mDNS module issues A and AAAA type queries and wait for a response.
nx_mdns_local_cache_clear	nx_mdns_local_cache_clear(&my_mdns); Clears all entries in the local service cache after sending the Goodbye message.
nx_mdns_peer_cache_clear	nx_mdns_peer_cache_clear(&my_mdns); Clears all entries in the peer service cache.

Note

For details on operation and definitions for the functions, data structures, typedefs, defines, API data, API structures, and function variables, review the associated Azure RTOS NetX Duo documentation in the References section.

Status Return Values

Name	Description
NX_MDNS_SUCCESS	mDNS success.
NX_MDNS_ERROR	mDNS internal error.
NX_MDNS_PARAM_ERROR	mDNS parameters error.
NX_MDNS_CACHE_ERROR	The cache size is not enough.
NX_MDNS_UNSUPPORTED_TYPE	The unsupported resource record type.
NX_MDNS_DATA_SIZE_ERROR	The data size is too big.
NX_MDNS_AUTH_ERROR	Attempting to parse too large data.
NX_MDNS_PACKET_ERROR	The packet can not add the resource record.
NX_MDNS_DEST_ADDRESS_ERROR	The destination address error.
NX_MDNS_UDP_PORT_ERROR	The UDP port error.

NX_MDNS_NOT_LOCAL_LINK	The message that not originate from the local link.
NX_MDNS_EXCEED_MAX_LABEL	The data exceed the max label size.
NX_MDNS_EXIST_UNIQUE_RR	At least one unique record in the cache.
NX_MDNS_EXIST_SHARED_RR	At least one shared record in the cache.
NX_MDNS_EXIST_SAME_QUERY	Exist the same query record in the cache.
NX_MDNS_EXIST_SAME_SERVICE	Exist the same service
NX_MDNS_NO_RR	No response for a one-shot query
NX_MDNS_NO_KNOWN_ANSWER	No known answer for the query
NX_MDNS_NAME_MISMATCH	The name mismatch
NX_MDNS_NOT_STARTED	mDNS does not start
NX_MDNS_HOST_NAME_ERROR	mDNS hostname error
NX_MDNS_NO_MORE_ENTRIES	No more entries are found
NX_MDNS_SERVICE_TYPE_MISMATCH	The service type mismatch
NX_MDNS_NOT_ENABLED	mDNS not enabled
NX_MDNS_ALREADY_ENABLED	mDNS already enabled

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.
Refer to the Azure RTOS NetX Duo documentation for additional information on mDNS/DNS-SD Module API functions.

4.3.37.3 NetX Duo mDNS/DNS-SD Module Operational Overview

NetX Duo mDNS/DNS-SD module manages two internal service caches: the local service cache, and the peer service cache.

Local service cache:

The local service cache stores Resource Records related to services offered by applications running on the node. For incoming queries, if the query matches the service offered, mDNS acknowledges with responses stored in the local service cache. Applications register services by calling the API nx_mdns_service_add(). To remove services, applications use the API nx_mdns_service_delete(), which will, in turn, send "goodbye" messages before removing the corresponding entries in the local service cache.

When a service is added, mDNS maintains at least 3 Resource Records in the local service cache: SRV, PTR, and TXT. Additional PTR Resource Record may be added if the service type includes subtype.

For example, an application registers a service:

name._*subtype*._sub._*type*._tcp.local

two PTR Resource Records are added to the local service cache, one for

"*_subtype._*sub*._type._*tcp.local *PTR name.type._*tcp*.*local"

and the other one for

*"_type._*tcp*.*local *PTR name.type._*tcp*.*local"

Peer service cache:

The peer service cache contains mDNS Resource Records received from neighboring nodes. mDNS module collects Resource Records advertised by other nodes on the network, and stores the received information in the peer service cache. When an application queries for information such as host IPv4 or IPv6 addresses, mDNS searches the peer service cache for locally cached responses. When an application queries for services offered by peers, mDNS searches through the cache for related PTR, SRV, TXT, and IPv4/IPv6 address records. The peer service cache also stores queries sent to the node. For example, an application may request a particular service by calling nx_mdns_service_one_shot_query. If the service is not found in the peer service cache, mDNS creates query questions (PTR, SRV, and TXT) in the peer service cache. These query questions will be sent to the network periodically till the service is resolved, or times out. Similarly, the application may use the API nx_mdns_service_continous_query() to request a particular service over a long period of time (application cancels a previously issued continuous query by using the API nx_mdns_service_query_stop()'). To search for a particular service in the peer service cache without sending queries to the network, applications can use the API nx_mdns_serivce_lookup(). This API only searches the resource records in the peer service cache.

Resource Record:

Each Resource Record is stored in a data structure NX_MDNS_RR in the service caches. Strings in Resource Records are of variable length, therefore are not stored in the NX_MDNS_RR structure. The Resource Record contains a pointer to the actual memory location where the string is stored. The string table and the Resource Records share the service cache. Resource Records are stored from the beginning of the service cache, and grow towards the end of the cache. The string table starts from the end of the service cache and grows towards the beginning of the cache. Each string in the string table has a length field and a counter field. When a string is added to the string table, if the same string is already present in the table, the counter value is incremented and no memory is allocated for the string. The service cache is considered full if no more resource records or new strings can be added to the service cache.

There are two ways for an application to find services offered on the local network. It can either issue a specific service look-up through a one-shot query, or it can initiate a continuous query to "monitor" the activities on the network.

One-Shot query:

In the one-short query scenario, the application must specify the service type. mDNS searches through the local service cache and the peer service cache. If a service instance is located, the one-shot query returns with the information found in the Resource Records. If there are no records in the local service cache or peer service cache, mDNS sends out query messages. If the instance name is specified, an ANY type (query the SRV and TXT type) with the specific instance name, in the form of name.type.local, is sent to the local network. If the instance name is not specified, a PTR type of query is sent to the local network. The first complete service received is returned to the caller.

Continuous query:

Continuous query works differently. The typical use case for a continuous query is to monitor the local network for a specific service (for example to constantly look for printing services on the local network). In this case, the application issues a search query (via the API nx_mdns_service_continious_query) for a certain type of service. The caller typically does not wait for a particular response. For queries submitted as continuous queries, the mDNS module transmits the queries periodically with exponentially increasing intervals. To stop the query, an application must use the API nx_mdns_service_query_stop to stop the internal timer in these queries. The query type can be NULL, in which case the query type is set to special PTR type _services._dns-sd._udp.local. This service type is defined by mDNS as a way to discover all services available on the local network. If the instance name is supplied, an ANY type (query the SRV and TXT type) with the specific instance name name.type.local is sent to the local network. If the instance name is NULL, a PTR type of query is sent to the local network, All responses, including responses from unsolicited queries, are recorded in the peer service cache. At a later time, the application uses the API nx_mdns_service_lookup to retrieve specific services from the peer service cache.

NetX Duo mDNS/DNS-SD Important Operational Notes and Limitations

NetX Duo mDNS/DNS-SD Module Operational Notes

The NetX Duo mDNS/DNS-SD requires a packet pool for transmitting mDNS messages; by default, the application must set the packet pool before using mDNS services. This can either be the packet pool used by the IP instance (g_packet_pool0), or it can be a separate packet pool added to the project: Azure RTOS -> NetX Duo -> NetX Duo Packet Pool Instance in NetX Duo mDNS Client.

When mDNS client sends a multicast message/query into the network asking which network participant matches with the hostname, the request goes into the all the participants in the network. The device in the network which matches the request responds to the entire network via multicast. All the participants are informed of the connection between the name and IP address, and can make a corresponding entry in their mDNS cache.

NetX Duo mDNS/DNS-SD Module Limitations

- When using NetX Duo secondary interface feature with mDNS, always use NetX Duo Source. NetX Duo library doesn't support secondary interface feature.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.37.4 Including the NetX Duo mDNS/DNS-SD Module in an Application

This section describes how to include NetX Duo mDNS/DNS-SD module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo mDNS/DNS-SD module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

NetX Duo mDNS/DNS-SD Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_mdns0NetXDuo mDNS/DNS-SD	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetXDuo mDNS/DNS-SD

When the NetX Duo mDNS/DNS-SD module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

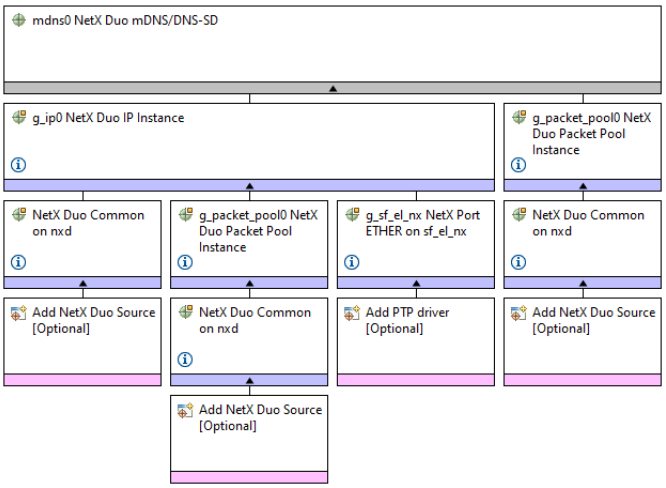


Figure 523: NetX Duo mDNS/DNS-SD Module Stack

In the stack above, the NetX Network Driver (or NetX Duo Network Driver in a NetX Duo stack) has not been populated yet. There are multiple possible selections for the Network Driver; they are all not provided so as not to needlessly complicate the figure and the following configuration tables. The available options depend on the MCU target, but some typical options include:

- NetX Duo Port using PPP on nxd_ppp
- NetX Port ETHER on sf_el_nx
- NetX Port using Cellular Framework on sf_cellular_nsal_nx
- NetX Port using PPP on nx_ppp
- NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.3.37.5 Configuring the NetX Duo mDNS/DNS-SD Module

The NetX Duo mDNS/DNS-SD module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-

accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo mDNS/DNS-SD Module

ISDE Property	Value	Description
mDNS/DNS-SD Server Support	Enable, Disable Default: Disable	This enables/disables mDNS/DNS-SD server functionality. Without the server functionality, the mDNS/DNS-SD module does not announce services provided by localhost, nor does it respond to mDNS inquiries.
mDNS/DNS-SD Client Support	Enable, Disable Default: Enable	This enables/disables mDNS/DNS-SD client functionality. Without the client functionality, mDNS/DNS-SD does not send queries, nor does it maintain mDNS query responses received over the network.
Validate address from Received mDNS messages	Enable, Disable Default: Enable	Validates addresses (source address, a destination address, and port numbers) from the received mDNS messages.
Verify multicast queries	Enable, Disable Default: Enable	Passive Observation Of Failures, mDNS /DNS_SD client (querier) observes the multicast queries issued by the other hosts on the network.
Enable mDNS/DNS-SD generating negative response	Enable, Disable Default: Enable	mDNS /DNS-SD server generates negative responses to queries for which it has legitimate ownership
Enable mDNS/DNS-SD IPV6 processing	Enable, Disable Default: Disable	Send/process mDNS message over IPv6 address.
Max IPV6 address count	2	Maximum IPv6 addresses count of the host.

Max string size for host	64	Maximum string size for the host name. *Note* *Does not include the NULL terminator.*
Max string size for service	64	Maximum string size for service name. *Note* *Does not include the NULL terminator.*
Max string size for Domain	16	Maximum string size for the domain name. *Note* *Does not include the NULL terminator.*
Max conflict count for host	8	Maximum conflict count for host name or service name
TTL value for resource records	120	TTL value for resource records with host name, in seconds.
TTL value for other records	4500	TTL value for other resource records, in seconds.
Time interval b/w mDNS probing messages	25	The time interval, in ticks, between mDNS probing messages.
Time interval b/w mDNS announcement messages	25	The time interval, in ticks, between mDNS announcement messages.
Time interval b/w goodbye messages	25	The time interval, in ticks, between repeated "goodbye" messages
Min time interval b/w two queries	100	The minimum time interval, in ticks, between two queries.
Max time interval b/w two queries	360000	The maximum time interval, in ticks, between two queries.
Min delay for sending the first query	2	The minimum delay for sending first query, in ticks.
Delay range for sending the first query	10	The delay range for sending first query, in ticks.

Query response time interval	100	The time interval, in ticks, in responding to a query to ensure an interval of at least 1sec since the last time the record was multicast
Probe query response time	25	The time interval, in ticks, in responding to a probe queries to ensure an interval of at least 250ms since the last time the record was multicast.
Unique query response delay	1	The delay, in ticks, in responding to a query to a service that is unique to the local network.
Minimum delay in responding to a query to a shared resource	2	The minimum delay, in ticks, in responding to a query to a shared resource.
Delay range in responding to a query to a shared resource	10	The delay range, in ticks, in responding to a query to a shared resource.
Minimum delay in responding to a query with TC bit	40	The minimum delay, in ticks, in responding to a query with TC bit.
Delay range in responding to a query with TC bit	10	The delay range, in ticks, in responding to a query with TC bit.
Timer count range	12	When sending out mDNS responses, the packet contains responses that otherwise would be sent within this timer counter range. The timer count range is expressed in ticks.
Number of retransmitted probing messages	3	The number of retransmitted probing messages.
Number of retransmitted goodbye messages	1	The number of retransmitted "goodbye" messages.
Minimum number of the count with no multicast response	2	The number of queries that no multicast response, then the host may take this as an indication that the record may no longer be valid.
Time interval in deleting the record from cache	1000	The time interval, in ticks, in deleting the record from the cache after seeing two or more of these queries, and seeing no multicast response containing the expected answer.

Delay for deleting a resource record	100	The delay for deleting a resource record when the TTL of this record is zero, in ticks.
Name	g_mdns0	Name of the mDNS/DNS-SD instance.
mDNS thread priority	3	Priority of the mDNS thread.
Internal thread stack size	4096	Size of the stack area in bytes
Local service cache size	4096	Storage space for local registered services in bytes.
Peer service cache size	4096	Storage space for service information received in bytes.
Name of Probing notify callback function	probing_notify	Optional callback function invoked at the end of the probing operation. It notifies the application whether the host name (when enabling mDNS on a local interface), or the service name (after registering a service) is unique.
Name of generated initialization function	mdns_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the NetX Duo mDNS/DNS-SD Lower-Level Modules

Only a few number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name

IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP cache storage units	Bytes, Entries Default: Bytes	ARP cache storage units selection
ARP Cache cache Size (in Bytes or storage units)	520	ARP Cache Size in Bytes/Entries selection. Must be a multiple of 52 Bytes. Note: 1 Entry = 52 Bytes
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1568	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

NetX Duo mDNS/DNS-SD Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group using Ethernet. Other MCUs and other interfaces like Wi-Fi or cellular may have different default values and available configuration settings.

NetX Duo mDNS/DNS-SD Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent

table illustrates an example selection for the ETHERC pins.

Note

The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note

The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII Default: Disabled	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only Default: _A only	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.37.6 Using the NetX Duo mDNS/DNS-SD Module in an Application

The steps in using the NetX Duo mDNS/DNS-SD module in a typical application are:

1. Wait for the network link to be enabled by calling the nx_ip_status_check (or if your system has multiple network interfaces, call nx_ip_interface_status_check) with the

NX_IP_LINK_ENABLED option.
2. Set the cache notify callback using the nx_mdns_cache_notify_set API.
3. Enable the mDNS functionality using the nx_mdns_enable API.
4. If the Application is configured as mDNS server then
    a. Register local service using nx_mdns_service_add API by providing the service name, type, subtype, priority, weight, and port.
    b. Once the service is successfully registered then a user-defined probing_notify callback is invoked.
    c. If all the operation is completed delete the service using nx_mdns_service_delete API.
5. If the Application is configured as mDNS client then
    a. Set the service change callback function to listen the service using nx_mdns_service_notify_set API.
    b. Perform single shot or continuous query using nx_mdns_service_one_shot_query and nx_mdns_service_continuous_query API respectively.
    c. Perform Service Lookup for particular service type using nx_mdns_service_lookup API.

The following figure illustrates common steps in a typical operational flow diagram:

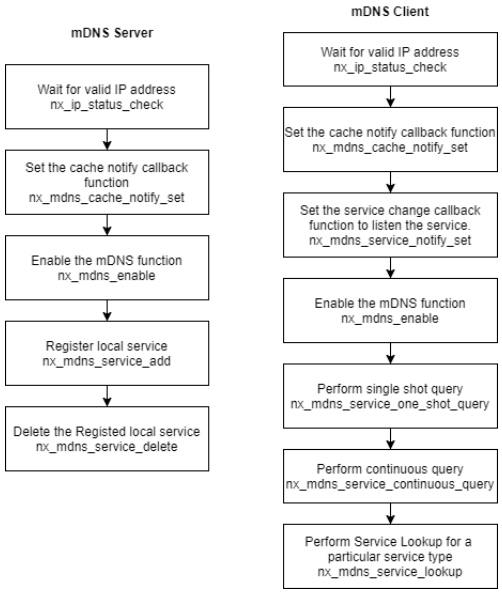


Figure 524: Flow Diagram of a Typical NetX Duo mDNS/DNS-SD Module Application

4.3.38 Azure RTOS USBX Overview

4.3.38.1 Azure RTOS USBX Interface Overview

The Azure RTOS USBX USB stack (ux) is integrated into the SSP. This document provides an overview and summary of the Azure RTOS USBX Interface for SSP. Specific USBX modules in SSP each have their own module overview section. Refer to the appropriate module overview sections in the SSP User's Manual for details on designing with specific modules.

4.3.38.2 What Does the Azure RTOS USBX Module Do?

USBX supports the two existing USB specifications: 1.1 and 2.0. It is designed to be scalable and will accommodate simple USB topologies with only one connected device, as well as complex topologies with multiple devices and cascading hubs. USBX supports both the host and device sides.

4.3.38.3 Supported USB Classes in Azure RTOS USBX

The USBX class modules listed below are fully supported in the Synergy Configuration tool. To use these USBX class modules, go to the Threads tab of the Synergy Configuration tool (configuration.xml), and select any of the USB class modules:

- ux_device_class_cdc_acm
- ux_device_class_hid
- ux_device_class_storage
- ux_host_class_cdc_acm
- ux_host_class_hid
- ux_host_class_printer
- ux_host_class_storage
- ux_host_class_video
- ux_host_class_hub

Note

Refer to the USBX Device Class User Guide, USBX Host Class User Guide or USBX Host Stack UVC User Guide for more information about the USBX Class specification. Each of the modules listed above also has an associated module overview section in the SSP User's Manual. Additionally, Module Guides for each of these modules are available from the Renesas Synergy web site.

The USBX class modules listed below are experimental modules for Synergy parts. The experimental modules are not currently supported in Synergy Configuration. To experiment with these USBX class modules, go to the Components tab of the Synergy Configuration tool (configuration.xml) and select any of the USB class modules:

- ux_device_class_cdc_ecm
- ux_device_class_rndis
- ux_host_class_audio
- ux_host_class_gser
- ux_host_class_prolific
- ux_host_class_swar
- ux_network_driver
- ux_device_class_pima
- ux_host_class_pima
- ux_pictbridge

Note

Any of the USB classes shown above are experimental and not yet tested for Synergy parts; it is not recommended to use them for product developments.

When the components above are added, a prebuilt library of the application code is also added. For each component listed above, there is an analogous component ending in '_src' that contains protected source files. The '_src' component can be added in addition to the prebuilt library module.

Writing an Application with USBX Modules

Refer to the associated module overview section in the SSP User's Manual for details on how to write

an application with USBX modules. Additionally, each of the associated Module Guides have a working application project that illustrates how to use the module in a typical application. Module Guides are available from the Renesas Synergy web site.

Writing an Application with Deprecated Modules

In the SSP version 1.2.0 or later, the USBX stack configuration is supported in the Synergy Configuration tool and the few USBX relevant components, which were defined in previous SSP versions, are now treated as [DEPRECATED] components. They are kept in the SSP version today to provide you with backward compatibility. Anyone using a new version of SSP and wanting to keep their existing application code compatible to an existing SSP version, can use [DEPRECATED] components.

In e^2 studio, create and configure a project and add the drivers:

1. Create the project: Creating a Project.
2. Configure the project: Configuring a Project.
3. Add the following components, if required, the same way as was done when using an existing version of SSP: Adding Drivers to a Thread and Configuring the Drivers.

Azure RTOS USBX Device Class CDC-ACM(option)	Threads	Framework > USB > [DEPRECATED] USBX Device Class CDC-ACM on ux_device_class_cdc_acm
Azure RTOS USBX (option)	Threads	Framework > USB > [DEPRECATED] USBX on ux

Note

Do not use components marked as [DEPRECATED] for new development. Only use these components for existing user applications which were developed with previous SSP releases.

USB Class Stack Configuration Overview

USBX Device Class Stack Configuration

The following figure shows the interface diagram of the USBX Device Class stack. The stack consists of one USBX device class component (ux_device_class_xxx) on the top, the USBX (ux) in the middle and the USBX Port driver (sf_el_ux Device Controller Driver (DCD)) on the bottom of the device class stack. As a recommended option, the SSP Transfer module (r_dmac) supports data transfer between memory and hardware FIFO in Synergy USB peripherals (USBHS or USBFS). To support the USB device stack configuration, there are components named USBX Device Configuration and USBX Interface Configuration. These two components do not represent actual software modules in SSP; they are virtual modules to handle the code generation.

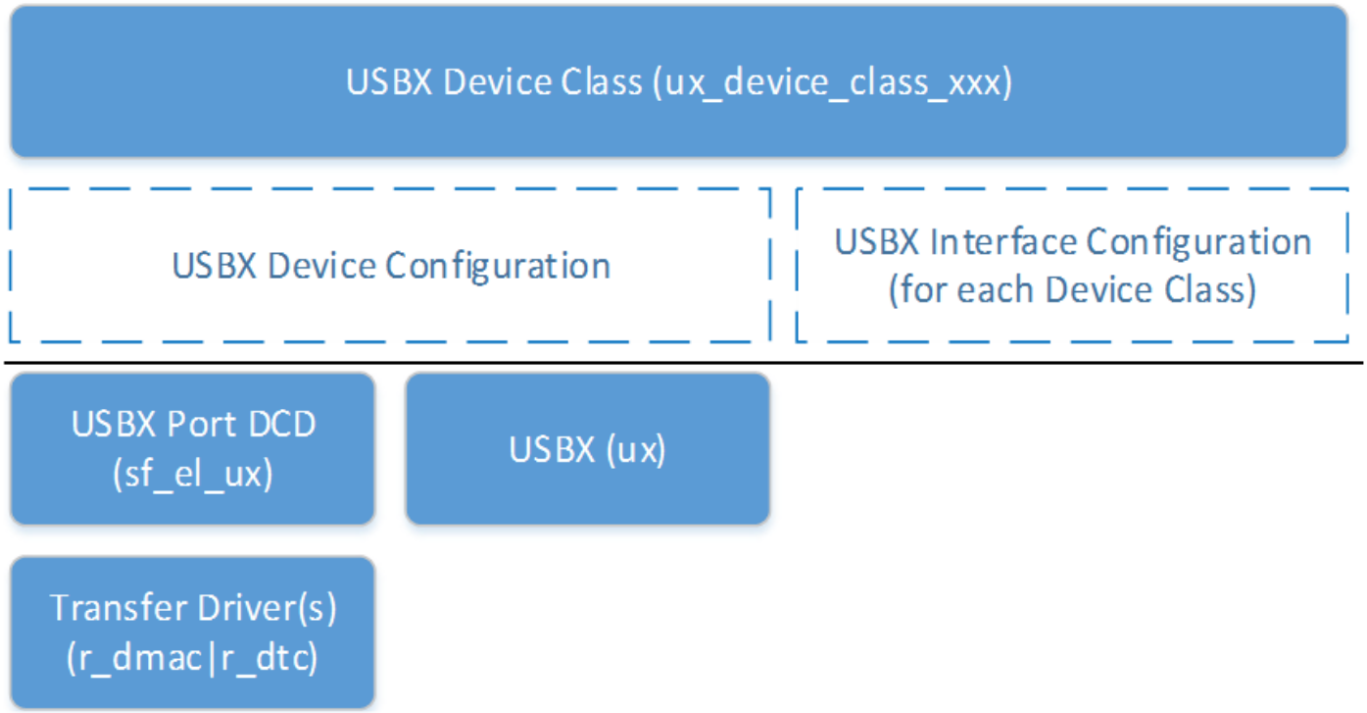


Figure 525: USBX Device Class Stack Configuration

Note: Currently, the r_dtc is not supported for the device side driver (only r_dmac is supported).

USBX Host Class Stack Configuration

The following figure shows the interface diagram of the USBX Host Class stack. The stack consists of one of the USBX Host Class components (ux_host_class_xxx) on the top, the USBX (ux) in the middle and the USBX Port driver component (sf_el_ux Host Controller Driver (HCD)) on the bottom of the host class stack. As a recommended option, the SSP Transfer module (r_dmac) supports data transfer between memory and hardware FIFO in Synergy USB peripherals (USBHS or USBFS). To support the USB host stack configuration, there is a virtual component (USBX Host Configuration) to handle the code generation:

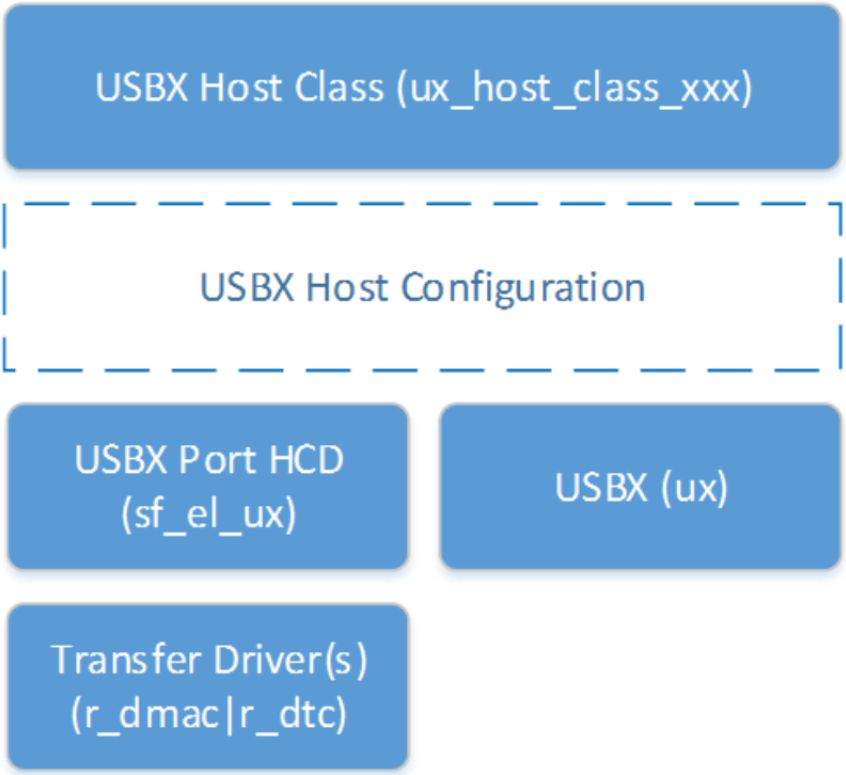


Figure 526: USBX Host Class Stack Configuration

Note

Currently, the r_dtc is not supported for the host side driver (only r_dmac is supported).

USB Device Descriptor Configuration

USBX Device Configuration

The USBX Device Configuration component has configurations, as shown in below table, to auto-generate the USB Device Descriptor and USB Configuration Descriptor. Refer to http://www.usb.org and download USB 2.0 Specification for more information about the specification of USB Device Descriptor and or Configuration Descriptor.

USBX Device Configuration

Configuration	Settings	Description
Composite Device	Enable or Disable (Default)	Enable this configuration only if composite device support is required in the application.
Vendor ID	16-bit arbitrary number (Default: 0x045B)	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (**idVendor**).
Product ID	16-bit arbitrary number (Default: 0x0000)	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (**idProduct**).

Device Release Number	16-bit arbitrary number (Default: 0x0000)	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (**bcdDevice**).
Index of Manufacturer String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iManufacturer**). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.
Index of Product String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iProduct**). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Index of Serial Number String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iSerialNumber**). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Device(0x00), Communications(CDC) (0x02, Default), HID (0x03), Mass Storage (0x08), Miscellaneous (0xEF), Vendor Specific (0xFF)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (**bDeviceClass**).
Index of String Descriptor describing this configuration	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (**iConfiguration**). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.

Size of USB Descriptor in bytes for this configuration	0x00 (Default) or value to be set to wTotalLength (calculated by user)	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (**wTotalLength**).
Number of Interfaces	0x00 (Default) or value to be set to bNumInterfaces (calculated by user).	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (**bNumInterfaces**).
Self-Powered	Enable (Default) or Disable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (**bmAttributes** bit6)
Remote Wakeup	Enable or Disable (Default)	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (**bmAttributes** bit5)
Maximum Power Consumption in 2mA units(0-250)	50 (Default), Integer value from 0 to 250.	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (**bMaxPower**).
Supported Language Code	16-bit number assigned by Manufacturer (Default: 0x0409)	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.

Name of USBX String Framework	Arbitrary C language symbol (Default: NULL)	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Total index number in USB String Descriptor in USBX String Framework	Arbitrary integer number (Default: 0)	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USB Language Descriptor	Arbitrary C language symbol (Default: NULL)	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.
Total Number of Language Support	Arbitrary integer number (Default: 0)	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.

USBX Device String Framework Configuration

The USBX String Framework is byte stream data to provide USB device information with human readable strings to the USB Host device. Users need to define the byte stream data in their application code if required. See the USBX Device Class User Guide section *Definition of the Strings of the Device Framework* for more information about the USBX String Framework. Here is an example of USBX String Framework, which consists of three indexes String descriptors.

```
const UCHAR g_usb_string_framework[] =
{
    /* Index #1 (this example shows manufacturer information) */
    (uint8_t) (0x0409), /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8), /* Byte1 Language Code */
    0x01, /* Byte2 Index */
    7, /* Byte3 Length */
```

```
    'R', 'E', 'N', 'E', 'S', 'A', 'S',
    /* Index #2 (this example shows product information) */
    (uint8_t) (0x0409), /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8), /* Byte1 Language Code */
    0x02, /* Byte2 Index */
    10, /* Byte3 Length */
    'C', 'O', 'M', ' ', 'D', 'E', 'V', 'I', 'C', 'E',
    /* Index #3 (this example shows Device Serial Number information) */
    (uint8_t) (0x0409), /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8), /* Byte1 Language Code */
    0x03, /* Byte2 Index */
    4, /* Byte3 Length */
    '0', '1', '0', '0'
};
```

You can configure the properties of the USBX Device Configuration component as shown in the following table. Refer to USBX Device Configuration for more information about each configuration in the table.

Example of the USBX String Framework Configuration

Configurations for String Descriptor on Synergy Configuration tool	Setting Example
Name of USB String Framework	g_usb_string_framework
Total index number of USB String Descriptor in USBX String Framework	3 (3 indexes)
Index of Manufacturer String Descriptor	1 (Index #1)
Index of Product String Descriptor	2 (Index #2)
Index of Serial Number String Descriptor	3 (Index #3)
Index of String Descriptor describing this configuration	0 (no string information)
Index of String Descriptor Describing Communications Class interface	0 (no string information)

USBX Device Language Framework Configuration

The USBX Languages Framework is byte stream data to support multiple languages. You need to define the byte stream data in your application code if required. See the USBX Device Class User Guide section *Definition of the Languages Supported by the Device for Each String* for more information. The following is an example of the USBX Language Framework which supports two

Languages:

```
const UCHAR g_usb_language_framework[] =
{
    /* US English */
    (uint8_t) (0x0409),
    (uint8_t) (0x0409 >> 8),
    /* Japanese */
    (uint8_t) (0x0411),
    (uint8_t) (0x0411 >> 8),
};
```

You can configure the properties of USBX Device Configuration component as shown in the following table. See section "USBX Device Configuration" for more information.

Example of the USBX Language Framework Configuration

Configurations for String Descriptor on Synergy Configuration tool	Setting Example
Name of USB Language Descriptor	g_usb_language_framework
Total Number of Language Support	2 (2 languages)

Device-Only Size Optimization

The Azure RTOS USBX module is built in device-only mode to reduce code size for Synergy S1 parts. The configuration (UX_SYSTEM_DEVICE_ONLY) is applied automatically if the S1 board is selected in the Synergy Configuration tool BSP tab. Note that the following configurations are fixed in the device-only mode.

- UX_THREAD_STACK_SIZE=512
- UX_SLAVE_REQUEST_DATA_MAX_LENGTH=512

USBX Hardware support details

	USBX Mass Storage Class				**USBX HID Class**			
	Host		**Device**		**Host**		**Device**	
	High Speed	**Full Speed**	**High Speed**	**Full Speed**	**High Speed**	**Full Speed**	**High Speed**	**Full Speed**
S1JA	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓
S124	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓

S128	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓
S3A1	N/A	✓	N/A	✓	N/A	✓	N/A	✓
S3A3	N/A	✓	N/A	✓	N/A	✓	N/A	✓
S3A6	N/A	☒	N/A	✓	N/A	☒	N/A	✓
S3A7	N/A	✓	N/A	✓	N/A	✓	N/A	✓
S5D5	N/A	✓	N/A	✓	N/A	✓	N/A	✓
S5D3	N/A	✓	N/A	✓	N/A	✓	N/A	✓
S5D9	✓	✓	✓	✓	✓	✓	✓	✓
S7G2	✓	✓	✓	✓	✓	✓	✓	✓
	USBX CDC/ACM				**USBX Video Class**		**USBX HUB Class**	
	Host		**Device**		**Host**		**Host**	
	High Speed	**Full Speed**	**High Speed**	**Full Speed**	**High Speed**	**Full Speed**	**High Speed**	**Full Speed**
S1JA	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓
S124	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓
S128	N/A	N/A	N/A	✓	N/A	N/A	N/A	✓
S3A1	N/A	✓	N/A	✓	N/A	☒	N/A	✓
S3A3	N/A	✓	N/A	✓	N/A	☒	N/A	✓
S3A6	N/A	✓	N/A	✓	N/A	☒	N/A	✓
S3A7	N/A	✓	N/A	✓	N/A	☒	N/A	✓
S5D5	N/A	✓	N/A	✓	N/A	✓	N/A	✓*
S5D3	N/A	✓	N/A	✓	N/A	☒	N/A	✓*
S5D9	✓	✓	✓	✓	✓	☒	✓	✓
S7G2	✓	✓	✓	✓	✓	☒	✓	✓

Symbol	Meaning
✓	Available (Tested)
☒	Not Available (Not tested/not functional or both)
N/A	Not supported by MCU

✓ * denotes that the HUB class is tested with self-powered hubs only.

4.3.38.4 Azure RTOS USBX Auto-generated Code Procedures

Note

Code samples in this section are subject to change. Note any warranty for contents and possible changes.

USBX Device Stack Auto-generated Code Procedures

The following code sample is pseudo code for USBX Device Classes. The function call sequence is auto-generated from the Synergy Configuration tool if one (or multiple) USBX Device Class component(s) is added to the Synergy Configuration tool. Auto-generated code is emitted to common_data.c file and provides the following features:

- Generates the USB Device Descriptor.
- Initializes the USBX software contexts in the memory pool.
- Initializes the USBX Device Stacks added in the Synergy Configuration tool.
- Initializes Synergy USB controller(s) in device mode. The USBX Device stack supports many-to-one topology between device classes and a device controller. For instance, two USBX Device Classes which consist of a USB composite device can use a single USB controller.

Note

The USBX Device stack does not support using multiple USB controllers simultaneously. Only one of the USB controllers is used at a time, even if the Synergy part has multiple USB controllers (like S7G2 parts).

```
SSP_VECTOR_DEFINE_UNIT();
void g_common_init(void)
{
    // Initialize USBX Memory.
    ux_system_initialize ();
    // Initialize USBX Device stack.
    ux_device_stack_initialize ();
    // Register the Device CDC-ACM Class if the class is used.
    ux_device_stack_class_register();
    // Register the Device HID Class if the class is used.
    ux_device_stack_class_register();
    // Register the Device Mass Storage Class if the class is used.
    ux_device_stack_class_register();
    // Initialize the USB Device Controller. This function calls either of
    // _ux_dcd_synergy_initalize() or
    // _ux_dcd_synergy_initalize_transfer_support()
    ux_dcd_initialize ();
}
```

USBX Host Stack Auto-generated Code Procedures

The following code sample is pseudo code for USBX Host Classes. The function call sequence is auto-generated from the Synergy Configuration tool if one (or multiple) USBX Host Class component(s) is added to the Synergy Configuration tool. Auto-generated code is emitted to the common_data.c file and provides the following features:

- Initializes the USBX software contexts in the memory pool.
- Initializes the USBX Host Stacks added in Synergy Configuration tool. Multi-instances of host class stacks are allowed to be built.
- Initializes the USBX HID Clients if the USBX Host HID Class is added in Synergy Configuration tool.
- Initializes Synergy USB controller(s) in host mode. The USBX Host stack supports Many-to-one or Many-to-many topology between host classes and host controllers. For instance, Two USBX Host Classes can use single USB controller, or Two USBX Host Classes can use different USB controllers individually if the Synergy part has multiple USB controllers (like S7G2 parts).
- Gets a USBX Host Class container for user application.

```
// Interrupt vector registering for USBHS or USBFS controller.
SSP_VECTOR_DEFINE_UNIT();
void g_common_init(void)
{
    // Initialize FileX (ONLY for Mass Storage Class)
    fx_system_initialize ();
    // Initialize USBX Memory.
    ux_system_initialize ();
    // Initialize the USBX Host stack.
    ux_host_stack_initialize ();
    // Register the HUB Class if the class is used.
    ux_host_stack_class_register();
    // Register the Host CDC-ACM Class if the class is used.
    ux_host_stack_class_register();
    // Register the Host HID Class if the class is used.
    ux_host_stack_class_register();
    // Register the Host HID Clents if the HID clients are used.
    ux_host_class_hid_clients_register ();
    // Register the Host Mass Storage Class if the class is used.
    ux_host_stack_class_register();
    // Register and initialize the USB Host Controller.
    ux_host_stack_hcd_register ();
    // Get the USBX Host Class Container for registered classes.
    ux_host_stack_class_get ();
}
```

4.3.38.5 Azure RTOS USBX Application Code Examples

Application projects are available in the module guides for each USBX module. Refer to the module guides for application projects showing working code for typical use cases.

4.3.38.6 Azure RTOS USBX Special Linker Sections

The USBX Device Stack configuration uses the following special memory sections in the linker script files. The order of memory sections in the linker script needs to consist of the USB Device Descriptor byte stream, which is given to _ux_device_stack_initialize() function; the linker script definitions must not be modified.

Memory section for the USBX Device Descriptor

Memory section	USB Descriptor to be defined in the section
.usb_device_desc_fs*	The USB Device Descriptor for FS mode
.usb_config_desc_fs*	The USB Configuration Descriptor for FS mode
.usb_interface_desc_fs*	The USB Interface Descriptor for FS mode
.usb_device_desc_hs*	The USB Device Descriptor for HS mode
.usb_config_desc_hs*	The USB Configuration Descriptor for HS mode
.usb_interface_desc_hs*	The USB Interface Descriptor for HS mode

Note

Memory sections for HS mode only exists for Synergy Parts, which has the USBHS controller.

4.3.38.7 Azure RTOS USBX Memory Requirements

The USBX Device stack and/or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generated code. The memory consumption is different for each class. Refer to the module overview section in the SSP User's Manual for the USBX memory requirements for a specific module.

4.3.38.8 Azure RTOS USBX Limitations

- Support for the USB Device vender-specific class is not available.
- The module needs the interrupt of a USB Controller enabled. See section "Logic USBX Synergy Port Framework Limitations" for more information.
- USBX classes (ux_device_class_cdc_ecm, ux_device_class_rndis, ux_host_class_audio, ux_host_class_gser, ux_host_class_prolific, ux_host_class_swar) and USBX network driver (ux_network_driver) are experimental modules and not yet tested for Synergy parts in this version of SSP. It is not recommended to use them for product developments.
- Composite device class mode is supported only on S3, S5 and S7 series (not on S1 series).
- Composite device class mode is tested with combination of CDC-CDC as well as CDC-MSC classes on Windows 10 PC acting as Host.
- In case of USBX composite device with DMA, MSC class will be given highest priority. for eg.
    - In case of CDC-MSC, MSC class uses DMA for data transfer and CDC class will use CPU for data transfer.
    - In case of CDC-CDC, one of the CDC class will use DMA for data transfer and other will use CPU for data transfer.
- In case of Device class CDC-ACM Read, there might be no difference between DMA and CPU

performance (Generally DMA should give better performance over CPU).
- When using Device class CDC-ACM Read, DMA will be used only when the requested transfer length is greater than the maximum packet size. For the transfer length less than maximum packet size, CPU will be used even if DMA module is added in ISDE configurator.

4.3.39 USBX Source

4.3.39.1 USBX Source Component Module Introduction

The purpose of this document is to provide an easy reference for the USBX source component in e^2 studio. The properties are explained in greater detail than the footer comment supplied with each property. Context specific usage is included for if and when to change a default value. This document should make it easier to use the USBX source component without having to cross reference with the Azure RTOS USBX User Guide, and help the developer get familiarized more quickly with USBX features.

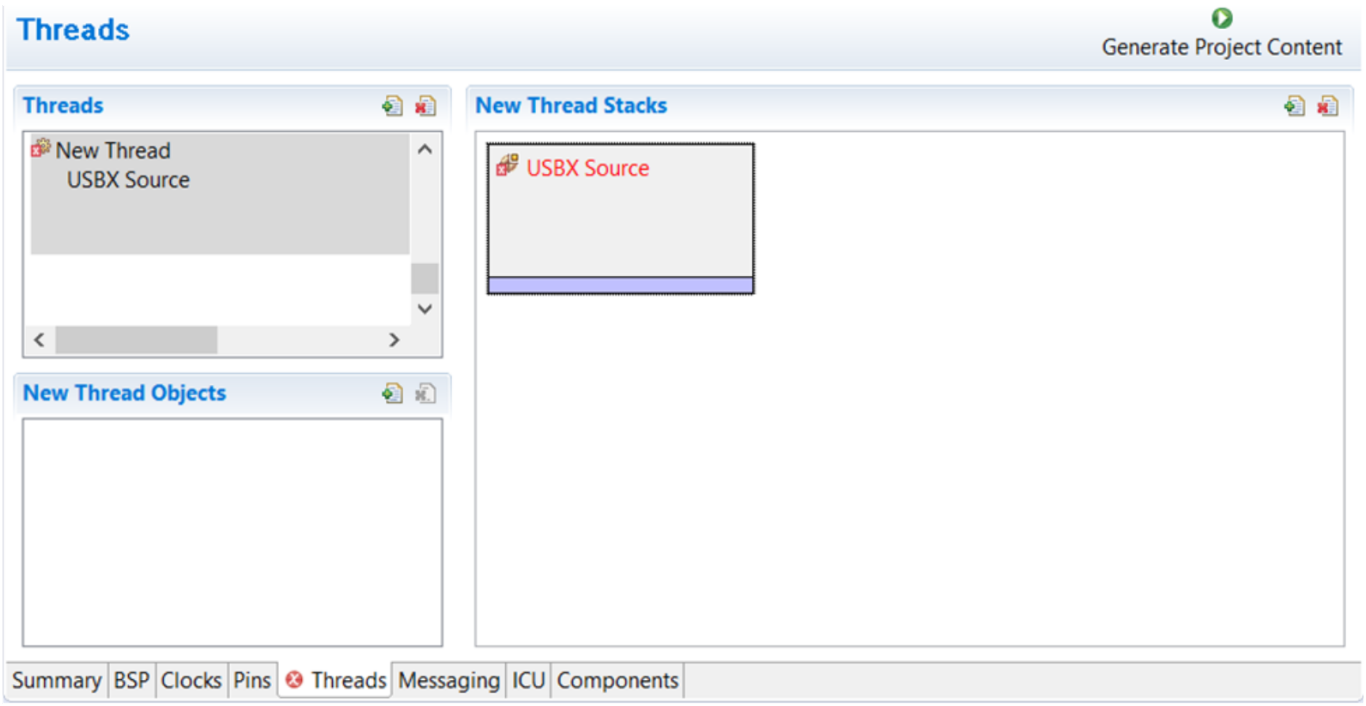
4.3.39.2 When to Include the USBX Source Component

Adding the USBX source component enables the developer in the Synergy configurator environment to customize the USBX library, change values from default settings and enable or disable certain features. Otherwise, they must use the prebuilt USBX library. In most projects beyond the simplest, the developer will typically want to customize their USBX environment. Adding USBX stack component does not add ThreadX stack source component by default.

Without adding the USBX source component, the e^2 studio configurator will use a prebuilt library with the USBX default settings.

4.3.39.3 Adding the USBX Source Component

In the e^2 studio configurator you can add the USBX Source component by selecting any thread from the Threads list and pressing the "New Stack" button and navigating the menu to X-Ware -> USBX -> Common -> USBX Source.



4.3.39.4 Changing the USBX Source Component Properties

After changing USBX property settings, the developer must click on the Generate Project Content button to update the project configurator in e^2 studio; then the USBX library must be rebuilt (for example, rebuild the project). Simply changing a property (or applying a #define in the preprocessor list) without rebuilding the project will not affect any change; e^2 studio will use the previously built library.

Default settings are based on use experience and are often the choice that will apply to the most common use cases.

4.3.39.5 USBX Source Component Overview

The properties of the USBX Source component are given in the order they appear in the properties window of the Synergy configurator.

Ticks per second for USBX system – default value not displayed, the ThreadX value is used – By default, USBX will use the value defined at ThreadX. You should not change this unless you have extensively modified the ThreadX tick timer mechanism.

Maximum Classes – default value not displayed, 8 used – When defined, this value is the maximum number of classes that can be loaded by USBX. This value represents the class container and not the number of instances of a class. For instance, if a particular implementation of USBX needs the hub class, the printer class and the storage class, then the UX_MAX_CLASSES value can be set to 3 regardless of the number of devices that belong to these.

Maximum Slave Classes – default value not displayed, 3 used – When defined, this value is the maximum number of classes in the device stack that can be loaded by USBX.

Maximum Slave Interfaces - default value not displayed, 16 used – When defined, this value

is the maximum number of interfaces in the device framework.

Maximum Host Class Containers - default value not displayed, no maximum set.

Maximum Device Class Containers - default value not displayed, no maximum set.

Maximum Host Controllers - default value not displayed, no maximum set - This value represents the number of different host controllers that are available in the system. For USB 1.1 support, this value will mostly be 1. For USB 2.0 support, this value can be more than 1. This value represents the number of concurrent host controllers running at the same time. If, for instance, there are two instances of OHCI running or one EHCI and one OHCI controllers running, the UX_MAX_HCD should be set to 2.

Maximum Devices - default value not displayed, 8 used – This value represents the maximum number of devices that can be attached to the USB. Normally, the theoretical maximum number on a single USB is 127 devices. This value can be scaled down to conserve memory. It should be noted that this value represents the total number of devices regardless of the number of USB buses in the system.

Maximum EDs - default value not displayed, 80 used – This value represents the maximum number of EDs in the controller pool. This number is assigned to one controller only. If multiple instances of controllers are present, this value is used by each individual controller.

Maximum TDs - default value not displayed, 128 used – This value represents the maximum number of regular TDs in the controller pool. This number is assigned to one controller only. If multiple instances of controllers are present, this value is used by each individual controller.

Note: Sufficient numbers of TDs are required, when user changes this parameter for large size (FS more than 8KB and HS more the 64KB) of data transfer. Sufficient number of 'Maximum TDs' needs to be specified in USBX Source module (USBX Source property >> Common >> Maximum TDs).

- Maximum TDs for FS mode (maximum data transfer size in bytes /64 bytes = TDs + additional TDs for SCSI wrapper commands).
- Maximum TDs for HS mode (maximum data transfer size in bytes /512 bytes = TDs + additional TDs for SCSI wrapper commands).

Maximum Isochronous TDs - default value not displayed, 128 used – This value represents the maximum number of isochronous TDs in the controller pool. This number is assigned to one controller only. If multiple instances of controllers are present, this value is used by each individual controller.

Stack size for USBX threads - default value not displayed, 1024 bytes used on host and mixed controllers, 512 on device only controllers – This value is the size of the stack in bytes for the USBX threads. It can be typically 1024 or 2048 bytes depending on the processor used and the host controller.

USBX Enumeration Thread Priority - default value not displayed, 20 used – This is the ThreadX priority value for the USBX enumeration threads that monitors the bus topology.

USBX Standard Thread Priority - default value not displayed, 20 used – This is the ThreadX priority value for the standard USBX threads.

USBX HID Keyboard Class Thread Priority - default value not displayed, 20 used – This is the ThreadX priority value for the USBX HID keyboard class.

USBX HCD Thread Priority - default value not displayed, 2 used – This is the ThreadX priority value for the host controller thread.

No use of time slice - default value disabled – If enabled the ThreadX target port does not use time slicing.

Maximum Slave Logical Units - default value not displayed, 2 used – This value represents the current number of SCSI logical units represented in the device storage class driver.

Maximum Host Logical Units - default value not displayed, 16 used – This value represents the maximum number of SCSI logical units represented in the host storage class driver.

Slave Request Control Maximum Length - default value not displayed, 256 used – This value represents the maximum number of bytes received on a control endpoint in the device stack. The default is 256 bytes, but can be reduced in memory constraint environments.

Note

Slave request control maximum length value have to increase more than event buffer value, If USBX Device HID class event buffer length value is more then 256 (USBX Device class HID source >> Common >> USBX Device HID Event Buffer Length).

Slave Request Data Maximum Length - default value not displayed, 512 in device only controllers, 4096 otherwise – This value represents the maximum number of bytes received on a bulk endpoint in the device stack. The default is 4096 bytes, but can be reduced in memory constraint environments.

Enforce Safe Alignment - default value is disabled – When enabled the memory allocation scheme enforces alignment. The default alignment value is UX_SAFE_ALIGN.

__Control Transfer Timeout - default value not displayed, 10000 used -__ When set, this value represents the control transfer timeout in milliseconds.

__Non-Control Transfer Timeout - default value not displayed, 50000 used -__ When set, this value represents the non-control transfer timeout in milliseconds.

__Disable CDC-ACM Non-blocking Transmission - default value is yes -__ Select yes to disable the CDC ACM non-blocking transmission support.

Note

To enable CDC-ACM Non-blocking transmission, USBX source and USBX device class CDC-ACM source should be added in thread stack and the property "Disable CDC ACM Non-blocking Transmission" in USBX source should be set to "No".

__Bidirectional Endpoint Support - default value is disabled -__ This property enables support for device bi-directional endpoint.

__Enable Assert Check - default value is yes -__ This property enables assert checks inside USBX.

__Enable Assertion On Failure Detection - default value is no -__ Select yes to take assert actions on failure detection.

__Host Device Class Code Validation - default value is disabled -__ When enabled, the host device class code is validated.

__USBX Standalone Mode - default value is disabled -__ When enabled, the standalone mode of USBX

is supported.

__USBX String Descriptor Requests With Zero Language ID - default value is disabled -__ When enabled, defines the processing of getting String Descriptor requests with zero Language ID.

__USBX Endpoint Buffer Owner - default value is disabled -__ When enabled, this value represents the endpoint buffer owner.

Show linkage warning - default value enabled – By default show linking warnings.

4.3.40 USBX Port

4.3.40.1 USBX Synergy Port Framework Introduction

The Azure RTOS USBX Synergy Port framework module (sf_el_ux) is integrated into the SSP and is used with Azure RTOS USBX. For more information about USBX (including API references), refer to the USBX User Guide.

Unsupported Features

The following features have not been integrated into this version of SSP:

- USBX host class asix
- USBX host class audio
- USBX host class gser
- USBX host class dpump
- USBX device class cdc-ecm
- USBX device class rndis
- USBX device class DFU
- USBX device class DPUMP
- USBX OTG host and device class

The following features are experimental. They have been integrated but not tested:

- USBX PIMA device class
- USBX Pictbridge device implementation

USBX Synergy Port Framework Module Features

The Azure RTOS USBX Synergy Port Framework module supports the following features:

- Implements Azure RTOS USBX in SSP- supports USBX APIs
- Supports the Port Device Controller Driver (DCD) for the USBHS peripheral
- Supports the Port Device Controller Driver (DCD) for the USBFS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBHS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBFS peripheral
- Supports manual switch over between host & device USB stack on same USB(ie USBHS or USBFS) port.
- Supports transfer module operation (optional)

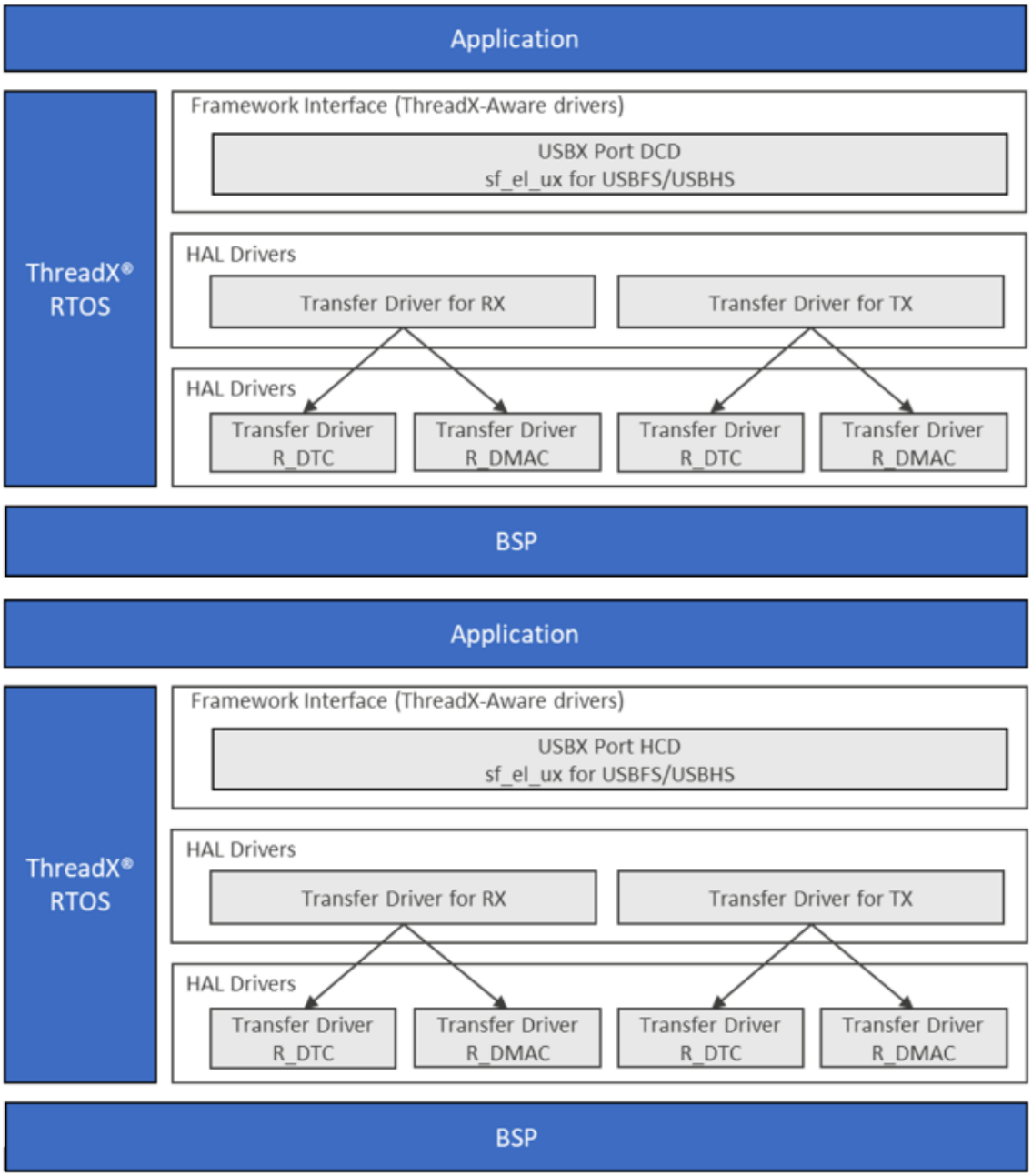


Figure 527: USBX Synergy Port Framework Module Block Diagram

—*

Note

*Currently, the r_dtc is not supported for both the host side driver and device side driver (only r_dmac is supported).****

4.3.40.2 USBX Synergy Port Framework Module APIs Overview

The Azure RTOS USBX Synergy Port Framework module doesn't have API calls of its own- it implements the API calls for the Azure RTOS USBX API calls. Documentation on these APIs is available in the Azure RTOS USBX User Manual.

4.3.40.3 USBX Synergy Port Framework Module Operational Overview

The Azure RTOS USBX Synergy Port framework module provides the Synergy USB hardware port functions required to use the USBX stack on Synergy hardware. Application code using this module is expected to use USBX API calls.

USBX Synergy Port Framework Module Important Operational Notes and Limitations

USBX Synergy Port Framework Module Operational Notes

The Azure RTOS USBX Synergy Port framework module includes the support for the Azure RTOS USBX APIs in SSP. Refer to the Azure RTOS USBX User Manual for a complete description of the available APIs.

The Azure RTOS USBX Synergy Port framework module supports the Port Device Controller Driver (DCD) on USBHS and USBFS peripherals as well as the Port Host Controller Driver (HCD) for the USBHS on USBFS peripherals.

Users have the option of using the Transfer Module for the USBX Synergy Port framework module to get better USB data throughput by transferring data in the block transfer mode. To enable the Transfer module, just add two instances of transfer components to the USBX Class stack in the Synergy Configuration tool and enable the interrupts in the property. The Synergy Configuration tool auto-generates the driver setup code to enable DMAC or DTC transfer in common_data.c.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

The module uses the interrupt of a USB Controller. Set the appropriate interrupt priority level in the Synergy Configuration tool; otherwise it does not work. The module uses the interrupt of a transfer module (implemented as DMAC or DTC) if it is used. Set the appropriate priority level in the Synergy Configuration tool. The priority level of the transfer module must be higher than the priority level for the USB Controller; otherwise it does not work.

USBX Synergy Port Framework Module Limitations

- Synergy USB controllers (USBHS and USBFS) have a limited number of PIPEs you can use for the isochronous transfer type (PIPE1 and PIPE2). This will limit the number of UVC devices (two devices) you can connect to the Synergy board configured as UVC HOST.
- The device side driver (sf_el_ux DCD driver) does not support DTC as the transfer interface.
- The host side driver (sf_el_ux HCD driver) does not support DTC as the transfer interface.
- The isochronous transfer is only supported for USB Host. The transfer type is not supported for USB Device.
- The USBFS controller is unlikely to support typical UVC devices. The maximum packet size of isochronous PIPEs on USBFS controller is limited to 256 bytes. This would impact in the UVC usage.
- Synergy USB controllers (USBHS and USBFS) do not support high-bandwidth isochronous transfer. The controllers support 1 transaction per micro-frame if running at high-speed).
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.40.4 Including the USBX Synergy Port Framework Module in an Application

This section describes how to include the USBX Synergy Port Framework module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Synergy Port Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Synergy Port Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_ux_hcd_hs_0 USBX Port HCD on sf_el_ux for USBHS	Threads	New Stack> X-Ware> USBX> Host > Synergy Port> USBX Port HCD on sf_el_ux for USBHS
g_sf_el_ux_hcd_fs_0 USBX Port HCD on sf_el_ux for USBFS	Threads	New Stack> X-Ware> USBX> Host > Synergy Port> USBX Port HCD on sf_el_ux for USBFS
g_sf_el_ux_dcd_hs_0 USBX Port HCD on sf_el_ux for USBHS	Threads	New Stack> X-Ware> USBX> Device > Synergy Port> USBX Port HCD on sf_el_ux for USBHS
g_sf_el_ux_dcd_fs_0 USBX Port HCD on sf_el_ux for USBFS	Threads	New Stack> X-Ware> USBX> Device > Synergy Port> USBX Port HCD on sf_el_ux for USBFS

When the USBX Synergy Port Framework module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

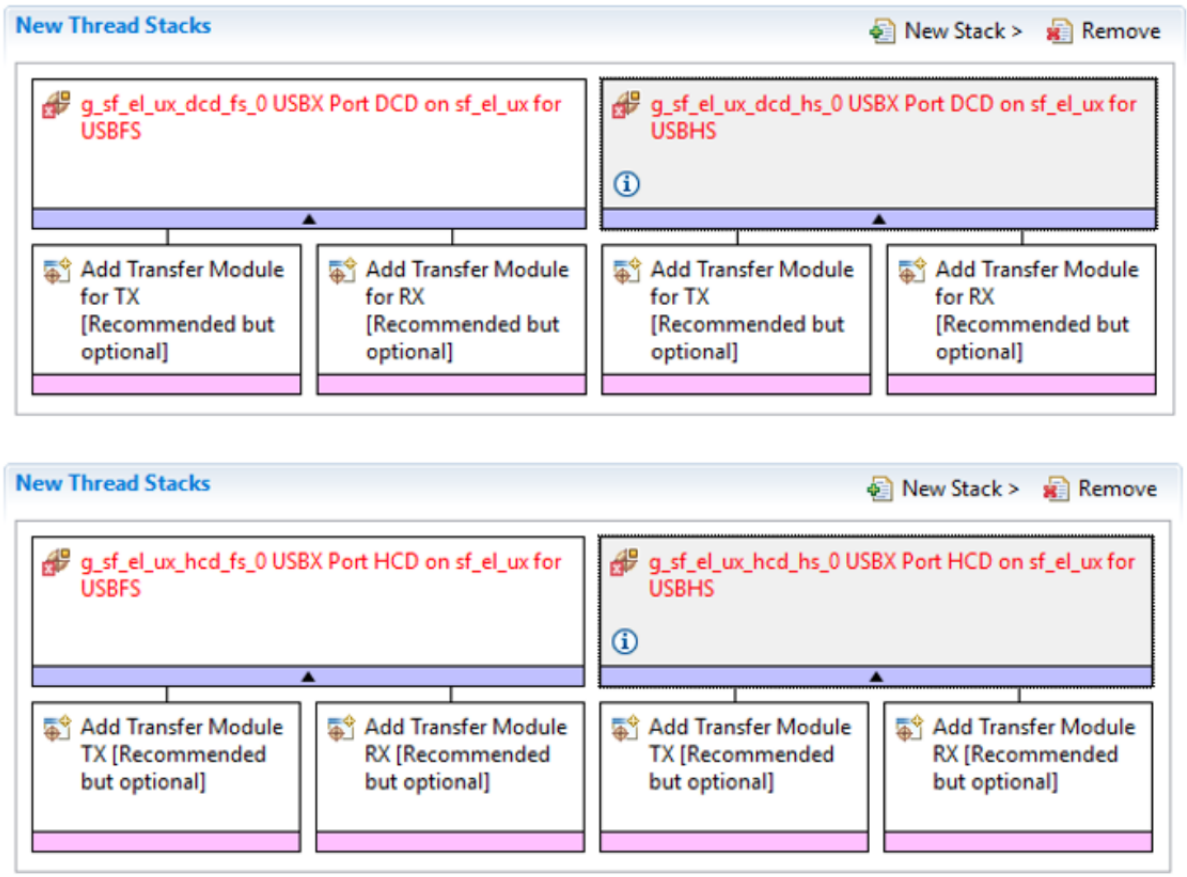


Figure 528: USBX Synergy Port Framework Module Stack

4.3.40.5 Configuring the USBX Synergy Port Framework Module

The USBX Synergy Port Framework module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description

Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.

Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0,1,2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Synergy Port Framework Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the Transfer Driver on r_dmac

ISDE Property	Value	Description

Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_transfer0	Module name.
Channel	0	Specify the hardware channel.
Mode	Block	Select the transfer mode.
Transfer Size	1 Byte	Select the transfer size.
Destination Address Mode	Fixed	Select the address mode for the destination.
Source Address Mode	Incremented	Select the address mode for the source.
Repeat Area (Unused in Normal Mode	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source	Device: Event USBFS FIFO 0 Host: Software Activation	Select the DMAC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback	NULL	A user callback that is called at the end of the transfer.
Transfer End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dmac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_transfer0	Module name.
Channel	0	Specify the hardware channel.
Mode	Block	Select the transfer mode.
Transfer Size	1 Byte	Select the transfer size.
Destination Address Mode	Incremented	Select the address mode for the destination.
Source Address Mode	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode	Destination	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	NULL	Specify the transfer destination pointer.
Source Pointer	NULL	Specify the transfer source pointer.
Number of Transfers	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	0	Specify the number of blocks to transfer in Repeat or Block mode.
Activation Source	Device: Event USBFS FIFO 1 Host: Software Activation	Select the DMAC transfer start event.
Auto Enable	False	Auto enable the transfer in open().
Callback	NULL	A user callback that is called at the end of the transfer.
Transfer End Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the transfer end interrupt priority.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Synergy Port Framework Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Synergy Port Framework Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table illustrates the pin selection method within the SSP Configuration Window and the subsequent tables demonstrates the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals are available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin

VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.40.6 Using the USBX Synergy Port Framework Module in an Application

Once the Azure RTOS Synergy Port USBX framework module is added to a thread, the Azure RTOS APIs become available for use by higher-level modules. There is usually no need to use the Azure RTOS Synergy Port USBX framework module directly in application code. Refer to the Azure RTOS USBX User's Manual if you need to use the module stand-alone and need to access the associated APIs directly.

Following is the sequence of API's called successfully in a sequential order in an application for

completely deleting detected USB Device class,

!( )[media/sf_el_ux_FD.png)

4.3.41 USBX Device Class CDC-ACM

4.3.41.1 USBX Device Class CDC-ACM Module Introduction

The USBX™ Device Class CDC-ACM module provides a high-level API for USBX Device Class CDC-ACM module applications and uses the USB and data-transfer peripherals on the Synergy MCU. A user defined callback can be created to determine when the stack activates or deactivates the USB CDC-ACM class.

USBX Device Class CDC-ACM Module Features

The USB Device Class CDC-ACM module allows for a USB host-system to communicate with the device as a serial device. This class is based on the USB standard and is a subset of the CDC standard. The USBX Device Class CDC-ACM module includes the following key features:

- Support for USB Full Speed (USBFS) or USB High Speed (USBHS)
- Receive and transmit data-transfer drivers for improved performance
- High-level APIs for reading and writing

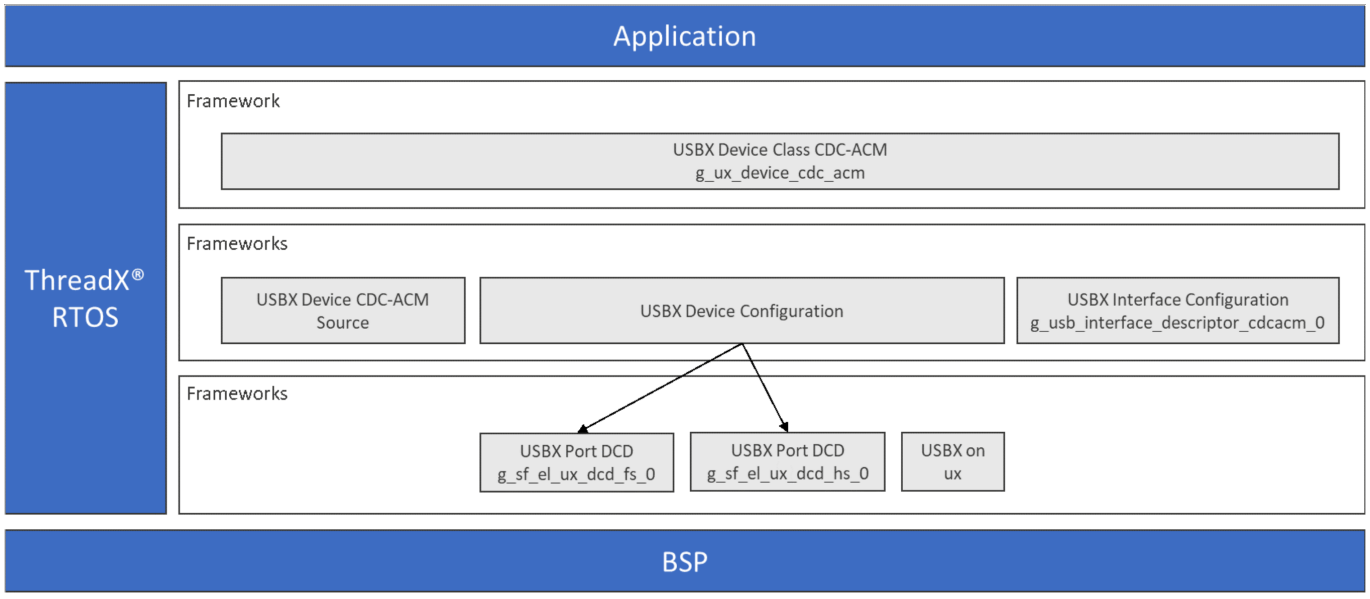


Figure 529: USBX Device Class CDC-ACM Module Block Diagram

4.3.41.2 USBX Device Class CDC-ACM Module APIs Overview

The USBX Device Class CDC-ACM Module defines APIs for reading and writing over the USB peripheral. A complete list of the available APIs, an example API call and a short description of each

can be found in the following table. A table of status return values follows the API summary table.

USBX Device Class CDC-ACM Module Summary

Function Name	Example API Call and Description
ux_device_class_cdc_acm_read	status = ux_device_class_cdc_acm_read(cdc, buffer, UX_DEMO_BUFFER_SIZE, &actual_length); This function is called when an application needs to read from the OUT data pipe (OUT from the host, IN from the device).
ux_device_class_cdc_acm_write	status = ux_device_class_cdc_acm_write(cdc, buffer, UX_DEMO_BUFFER_SIZE, &actual_length); This function is called when an application needs to write to the IN data pipe (IN from the host, OUT from the device).

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
UX_SUCCESS	This operation was successful.
UX_CONFIGURATION_HANDLE_UNKNOWN	Device is no longer in the configured state.
UX_TRANSFER_NO_ANSWER	No answer from device. The device was probably disconnected while the transfer was pending.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.41.3 USBX Device Class CDC-ACM Module Operational Overview

The initialization of the CDC-ACM class expects some specific parameters as illustrated in the application project associated with this module guide.

The CDC-ACM is based on a USB-IF standard and is automatically recognized by Mac OS® and Linux OS®. On Windows® platforms, this class requires a .inf file. Azure RTOS supplies a template for the CDC-ACM class, and it can be found in the usbx_windows_host_files directory. For more recent versions of Windows, the file CDC_ACM_Template_Win7_64bit.inf should be used; this file needs to be modified to reflect the PID/VID used by the device. The PID/VID will be specific to the final customer when the company and the product are registered with the USB-IF. In the .inf file, the fields to modify are described in the application project associated with this module guide.

In the device framework of the USBX CDC-ACM device, the PID/VID are stored in the device descriptor (see the device descriptor declared in the application project referenced in the preceding paragraph).

When a USB host-system discovers the USBX CDC-ACM device, it will mount a serial class and the

device can be used with any serial terminal program. (See the host operating system for reference.)

USBX Device Class CDC-ACM Module Important Operational Notes and Limitations

USBX Device Class CDC-ACM Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX Device CDC-ACM (ux_device_class_cdc_acm)	6.1KB	18KB

Note

The information shown in the table above is valid if compiled with default USBX configurations. The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 512 (bytes) for S1 parts; 2048 (bytes) for the other parts

The application needs to save the instance with the callback function registered in USBX CDC-ACM instance_activate Function Callback. Read and write are executed using the saved instance.

USBX Device Class CDC-ACM Module Limitations

- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.41.4 Including the USBX Device Class CDC-ACM Module in an Application

This section describes how to include the USBX Device Class CDC-ACM Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Device Class CDC-ACM Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Device Class CDC-ACM Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_device_class_cdc_acm0 USBX Device Class CDC-ACM	Threads	New Stack> X-Ware> USBX> Device> Classes > CDC-ACM > USBX Device Class CDC-ACM

When the USBX Device Class CDC-ACM Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

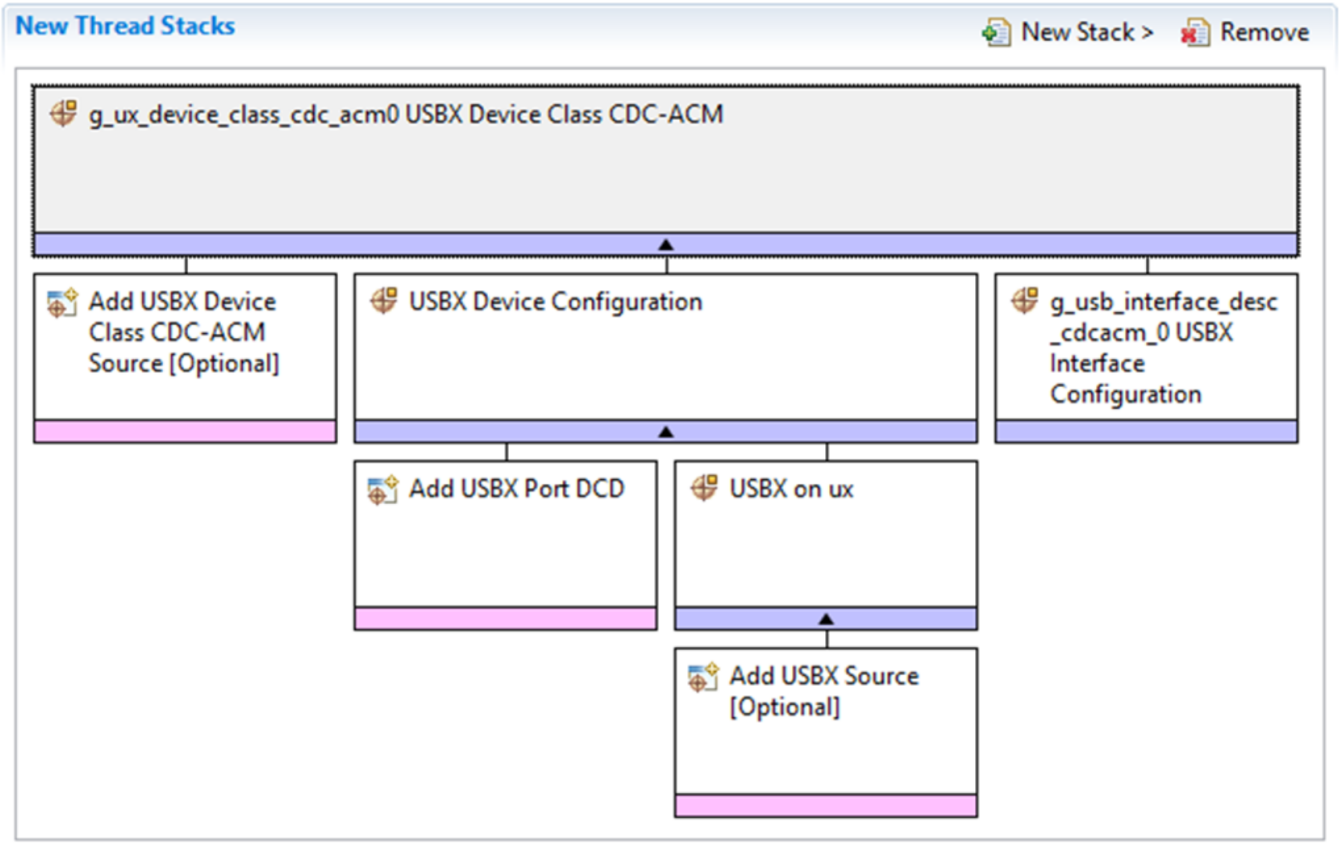


Figure 530: USBX Device Class CDC-ACM Module Stack

4.3.41.5 Configuring the USBX Device Class CDC-ACM Module

The USBX Device Class CDC-ACM Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Device Class CDC-ACM Module

ISDE Property	Value	Description
Name	g_ux_device_class_cdc_acm0	Specify the name of the USBX Device CDC-ACM Class module instance. It must be a valid C symbol.
USBX CDC-ACM instance_activate Function Callback	ux_cdc_device0_instance_activate	Specify the name of the instance_activate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class" for more information about the instance_activate callback function.
USBX CDC-ACM instance_deactivate Function Callback	ux-cdc_device0_instance_deactivate	Specify the name of the instance_deactivate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class" for more information about the instance_activate callback function.
USBX CDC-ACM parameter_change Function Callback	NULL	Specify the name of the parameter change user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class" for more information about the parameter_change callback function.
Name of generated initialization function	ux_device_class_cdc_acm_init0	Name of generated initialization function selection.

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Device Class CDC-ACM Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Device Class CDC-ACM Source

ISDE Property	Value	Description
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.
CDC ACM device write auto ZLP enable	Enabled, Disabled Default: Disabled	UX_DEVICE_CLASS_CDC_ACM_WRITE_AUTO_ZLP When enabled, the CDC class sends a ZLP automatically after a buffer is sent.
Device CDC ACM Zero Copy	Enabled, Disabled Default: Disabled	UX_DEVICE_CLASS_CDC_ACM_ZERO_COPY When enabled, device CDC ACM zero copy for bulk in/out endpoints is supported.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Device Configuration Instance

ISDE Property	Value	Description
Vendor ID	0x045B	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (idVendor).

Product ID	0x0000	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (idProduct).
Device Release Number	0x0000	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (bcdDevice).
Index of Manufacturing String Descriptor	0x00	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iManufacturer). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.
Index of Product String Descriptor	0x00	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iProduct). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Index of Serial Number String Descriptor	0x00	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iSerialNumber). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor specific Default: Communications(CDC)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (bDeviceClass).

Index of String Descriptor describing this configuration	0x00	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (iConfiguration). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (wTotalLength).
Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (bNumInterfaces).
Self-Powered	Enable, Disable Default: Enable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit6).
Remote Wakeup	Enable, Disable Default: Disable	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit5).
Maximum Power Consumption (in 2mA units)	50	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (bMaxPower).

Supported Language Code	0x0409	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.
Name of USBX String Framework	NULL	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Total index number of USB String Descriptors in USB String Framework	0	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USBX Language Framework	NULL	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.
Number of Languages to support (US English is applied if zero is set)	0	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.
Name of generated initialization function	ux_device_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Interface Configuration CDC-ACM Instance

ISDE Property	Value	Description
Name	g_usb_interface_desc_cdcacm_0	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.
Interface Number of Communications Class interface	0x00	Specify the index number of Communications Class interface. This configuration is a part of the USB Interface Descriptor (bInterface). The number must not be duplicated with the Interface Number of Data Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Interrupt Transfer endpoint to use for Communications Class	Endpoint 1-9 Default: Endpoint 3	Specify the Endpoint Number of Interrupt Endpoint. It must not be duplicated with ones for the other Endpoints.
Polling period for Interrupt Endpoint (in mS/125us units for FS/HS)	0x0F	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt Endpoint and ignored for Bulk Endpoints. Value is in frame counts (1ms units for FS mode and 125us units for HS mode).
Interface Number of Data Class interface	0x01	Specify the index number of Data Class interface. This configuration is a part of the USB Interface Descriptor (bInterface). The number must not be duplicated with the Interface Number of Communications Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Bulk In Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 1	Specify the Endpoint Number of Bulk In Endpoint. It must not be duplicated with ones for the other Endpoints.
Bulk Out Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 2	Specify the Endpoint Number of Bulk Out Endpoint. It must not be duplicated with ones for the other Endpoints.

Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface)	0x00	Specify the index number of String Descriptor Describing Communications Class interface. This configuration is a part of the USB Interface Descriptor (iInterface). Set '0' if do not have String information for the interface.
Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface)	0x00	Specify the index number of String Descriptor Describing Data Class interface. This configuration is a part of the USB Interface Descriptor (iInterface). Set '0' if do not have String information for the interface.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.

Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Device Class CDC-ACM Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Device Class CDC-ACM Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence

USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin

VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.41.6 Using the USBX Device Class CDC-ACM Module in an Application

The configurator generates processing to create and register the USBX Device Class CDC-ACM module; however, communication must be done after the device is connected to the host.

The typical steps in using the USBX Device Class CDC-ACM module in an application are:

1. Wait for the callback function registered in USBX CDC-ACM instance_activate Function Callback to be called.

2. In the callback function, save the instance.

3. For received data reading, use the ux_device_class_cdc_acm_read API.

4. For sending data, use the ux_device_class_cdc_acm_write API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

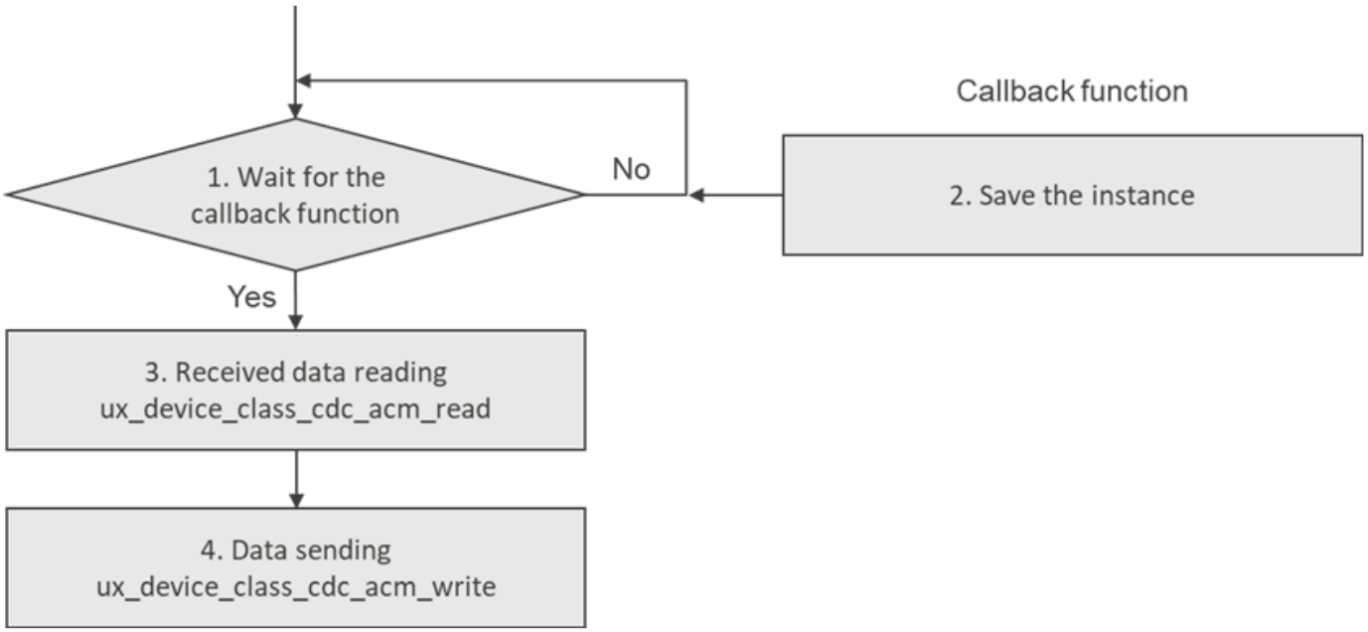


Figure 531: Flow Diagram of a Typical USBX Device Class CDC-ACM Module Application

4.3.42 USBX Device Class HID

4.3.42.1 USBX Device Class HID Module Introduction

The USBXTM Device Class Human Interface Design (HID) module provides a high-level API for HID applications and configures the USBX Device Class HID Source, USBX Host Configuration, USBX Source and USBX Port HCD. The USBX Device Class HID module uses the USB peripheral on the Synergy MCU.

Unsupported Features

USBX composite device class supports only CDC-CDC and CDC-MSC device class.

USBX Device Class HID Module Features

The USB Device Class HID module allows a USB host system to communicate with the device as a keyboard device, a mouse device and other HID devices. This class is based on the USB standard and is a subset of the HID standard. The USBX Device Class HID module includes the following key features:

- Support for USB high speed (USBHS) or full speed (USBFS)
- Uses Receive and Transmit data-transfer drivers for improved performance
- Provides high-level APIs for reading and writing

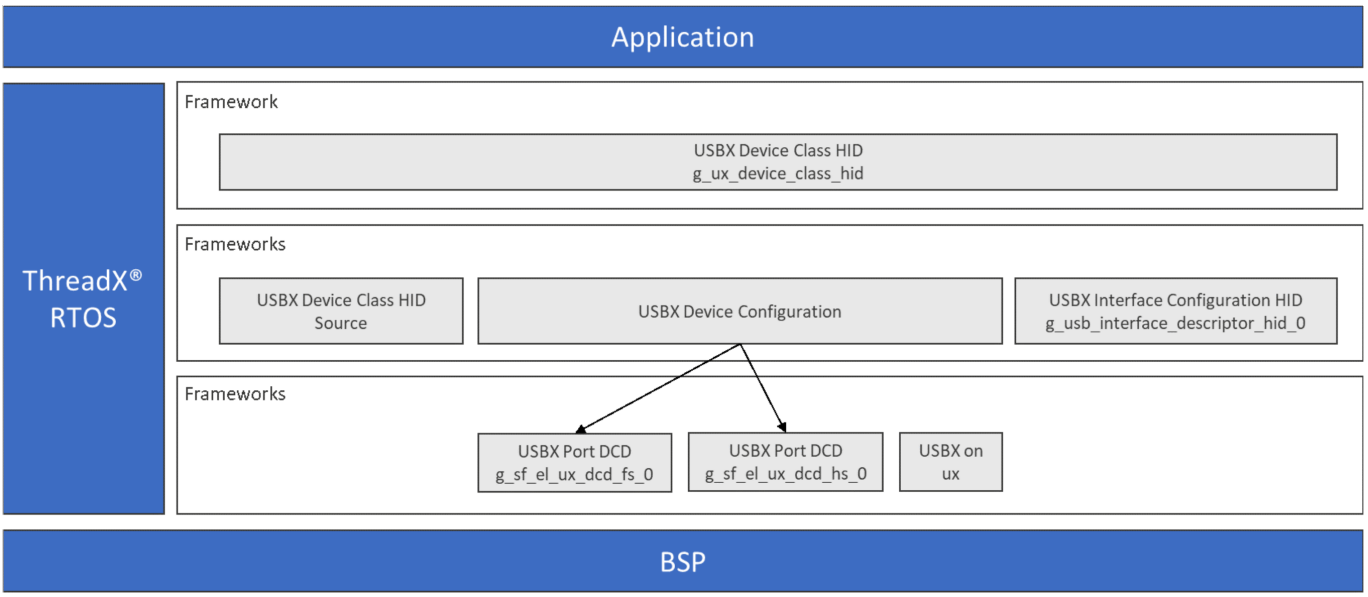


Figure 532: USBX Device Class HID Module Block Diagram

4.3.42.2 USBX Device Class HID Module APIs Overview

The USBX Device Class HID module defines APIs for sending and receiving HID events and reports. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

USBX Device Class HID Module Summary

Function Name	Example API Call and Description
ux_device_class_hid_event_set	ux_device_class_hid_event_set (hid, &hid_event); This function is called when an application needs to send a HID event to the host.
ux_device_class_hid_event_get	ux_device_class_hid_event_get (hid, & hid_event); This function is called when an application needs to receive a HID event from the host.
ux_device_class_hid_report_set	ux_device_class_hid_report_set (hid, descriptor_type, request_index, host_length); This function is called when an application needs to send a HID report to the host.
ux_device_class_hid_report_get	ux_device_class_hid_report_get (hid, descriptor_type, request_index, host_length); This function is called when an application needs to receive a HID report to the host.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
UX_SUCCESS	The data transfer was completed.
UX_TRANSFER_TIMEOUT	Transfer timeout, reading/writing not completed.
UX_MEMORY_INSUFFICIENT	Not enough memory.
UX_HOST_CLASS_UNKNOWN	Wrong class instance.
UX_FUNCTION_NOT_SUPPORTED	Unknown IOCTL function.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.42.3 USBX Device Class HID Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Device Class HID Module Important Operational Notes and Limitations

USBX Device Class HID Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX Device HID (ux_device_class_hid)	6.1KB + (Add the additional memory as per Note[*1]).	12KB + (Add the additional memory as per Note[*1]).

Note

*[*1]: If maximum number of HID events queue is more than 16(USBX device class HID source module: Property >> Common >> Maximum number of USBX device HID event queue) or HID event buffer length is more than 64(USBX device class HID source module: Property >> Common >> USBX Device HID Event length Buffer) in the XML configurator. Add the additional memory to USBX memory pool as follows:*

- Additional memory calculation for USBX full speed and High speed made.
  - ((Maximum number of USBX device HID event queue - 16) * 12) + ((Maximum number of USBX device HID event queue * USBX Device HID Event length Buffer) - 1024) = Additional memory in bytes.

Note: If HID event buffer value is increased to large value, Application stack size and USBX thread stack size(USBX Source >> Common >> Stack size for USBX threads) need to increase.

Note

The information shown in the table above is valid if compiled with default USBX configurations.
The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 512 (bytes) for S1 parts; 2048 (bytes) for the other parts

- The application gets the HID instance of the slave device from the global variable, _ux_system_slave; sends and receives executed using this instance.
- Use the Protocol code property of the USBX Interface Configuration HID Driver to determine

the operation of the actual device.

USBX Device Class HID Module Limitations

- The module needs the interrupt of a USB Controller enabled.
- The module uses the interrupt of a USB Controller. Set the appropriate interrupt-priority level in the Synergy Configuration tool for proper operation.
- The module uses the interrupt of a transfer module (implemented as DMAC or DTC) if it is used. Set the appropriate priority level in the Synergy Configuration tool and the level has to be higher than a USB Controller, otherwise it does not work.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

Note

Currently, DTC is not supported by the device side driver (only DMAC is supported).

4.3.42.4 Including the USBX Device Class HID Module in an Application

This section describes how to include the USBX Device Class HID Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Device Class HID Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Device Class HID Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_device_class_hid USBX Device Class HID	Threads	New Stack> X-Ware> USBX> Device > Classes > HID > USBX Device Class HID

When the USBX Device Class HID Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

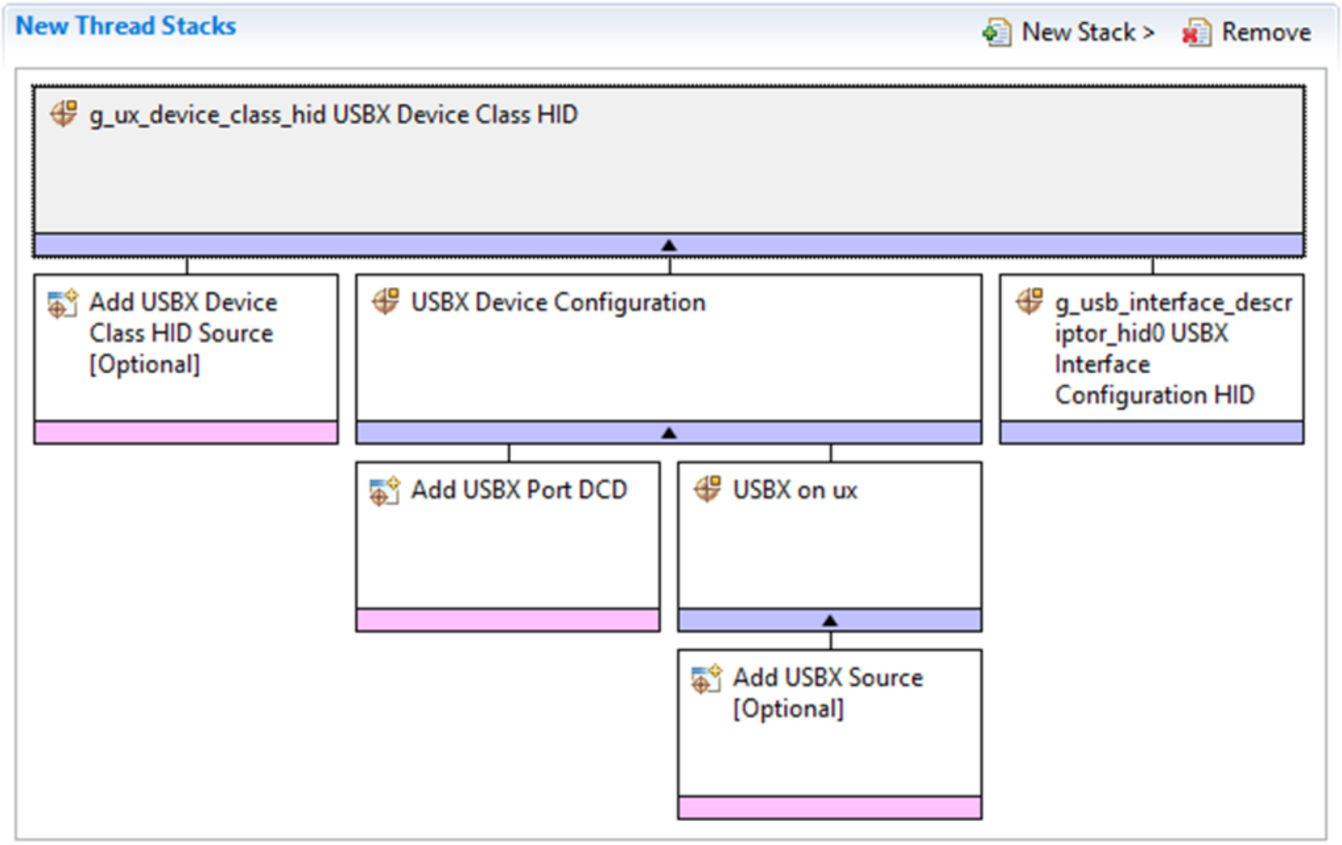


Figure 533: USBX Device Class HID Module Stack

4.3.42.5 Configuring the USBX Device Class HID Module

The USBX Device Class HID Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Device Class HID Module

ISDE Property	Value	Description

Name	g_ux_device_class_hid	Specify the name of USBX Interface Descriptor for HID Class. It must be a valid C symbol.
USBX Device HID Entry Function	ux_device_class_hid_entry	Specify the name of a user callback function to get an event from HID Class. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device HID Class" for more information about the user callback function.
USBX Device HID User Callback Function	ux_hid_device_callback	Specify the name of user entry function for the USBX Device HID Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device HID Class" for more information about the user entry function.
USBX Device HID GetReport Callback Function	NULL	Specify the name of user entry function for the USBX Device HID Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device HID Class" for more information about the user entry function.
USBX Device HID Instance Activate Callback Function	NULL	Specify the name of instance_activate user callback function for the USBX Device HID Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device HID Class" for more information about the instance_activate callback function.

USBX Device HID Instance Deactivate Callback Function	NULL	Specify the name of instance_deactivate user callback function for the USBX Device HID Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device HID Class" for more information about the instance_activate callback function.
Multiple HID Report Support	Enable, Disable Default: Disable	Set Enable to support multiple HID report. This configuration is used to indicate which data fields are represented in each report structure.
Name of generated initialization function	ux_device_class_hid_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Device Class HID Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Device Class HID Source

ISDE Property	Value	Description

USBX Device HID Event Buffer Length	Value must be greater than or equal to 32. Default : 64	UX_DEVICE_CLASS_HID_EVENT_BUFFER_LENGTH Defines the size of Event Buffer Length. if the event buffer length value is more than 256. Slave request control maximum length value have to increase more than Event buffer length value. in USBX source module. (USBX Source: Property >> common >> Slave Request Control Maximum Length).
Maximum number of USBX Device HID Event queue.	Value must be greater than or equal to 16. Default: 16.	'UX_DEVICE_CLASS_HID_MAX_EVENTS_QUEUE' This value represents the maximum number of hid event queue, in the USBX Device hid class driver.
Device HID Interrupt Out Support	Enabled, Disabled Default: Disabled	UX_DEVICE_CLASS_HID_INTERRUPT_OUT_SUPPORT When enabled, device HID interrupt OUT transfer is supported.
Device HID Zero Copy	Enabled, Disabled Default: Disabled	UX_DEVICE_CLASS_HID_ZERO_COPY Defined, it enables device HID zero copy and flexible queue support (works if HID owns endpoint buffer).
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Device Configuration Instance

ISDE Property	Value	Description
Vendor ID	0x045B	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (idVendor).

Product ID	0x0000	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (idProduct).
Device Release Number	0x0000	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (bcdDevice).
Index of Manufacturing String Descriptor	0x00	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iManufacturer). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.
Index of Product String Descriptor	0x00	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iProduct). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Index of Serial Number String Descriptor	0x00	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iSerialNumber). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor specific Default: Communications(CDC)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (bDeviceClass).

Index of String Descriptor describing this configuration	0x00	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (iConfiguration). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (wTotalLength).
Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (bNumInterfaces).
Self-Powered	Enable, Disable Default: Enable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit6).
Remote Wakeup	Enable, Disable Default: Disable	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit5).
Maximum Power Consumption (in 2mA units)	50	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (bMaxPower).

Supported Language Code	0x0409	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.
Name of USBX String Framework	NULL	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Total index number of USB String Descriptors in USB String Framework	0	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USBX Language Framework	NULL	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.
Number of Languages to support (US English is applied if zero is set)	0	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.
Name of generated initialization function	ux_device_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Interface Configuration HID Instance

ISDE Property	Value	Description
Name	g_usb_interface_descriptor_hid0	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.
Protocol code (None(0) /Keyboard(1) /Mouse(2)/Keyboard+Mouse(3))	0x00	Both the keyboard and mouse interface will be available only if Keyboard+Mouse(3) protocol is selected. Select device class as Device (0x00) in device configuration when protocol selected is (Keyboard+Mouse).
(Keyboard) Interface Number of HID Class interface	0x00	Keyboard Interface will be available for use when protocol code selected is either Keyboard(1) or Keyboard+Mouse(3).
(Keyboard) Endpoint Number to be used for Interrupt-In	Endpoint 1-9 Default: Endpoint 1	Specify the Endpoint Number of Interrupt-In Endpoint. It must not be duplicated with ones for the other Endpoints.
(Keyboard) Maximum packet size in bytes for Interrupt-In	0x8	Specify the maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
Interval for polling Interrupt-In EP for data transfers (milliseconds)	0x8	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt-In Endpoint. Value is in frame counts (1ms units for FS mode and 125us units for HS mode).
(Keyboard) Interrupt-Out Endpoint (Optional)	Enable, Disable Default: Disable	This configuration is reserved and currently not used.
Endpoint Number for Interrupt-Out (Optional)	Endpoint 1-9 Default: Endpoint 3	This configuration is reserved and currently not used.
(Keyboard) Maximum packet size in bytes for Interrupt-Out EP (Optional)	0x8	This configuration is reserved and currently not used.
(Keyboard) Interval for polling Interrupt-Out EP for data transfers (milliseconds) (Optional)	0x8	This configuration is reserved and currently not used.

(Mouse) Interface Number of HID Class interface	0x01	Mouse Interface will be available for use when protocol code selected is either Mouse(2) or Keyboard+Mouse(3).
(Mouse) Endpoint Number to be used for Interrupt-In	Endpoint 1-9 Default: Endpoint 2	Specify the Endpoint Number of Interrupt-In Endpoint. It must not be duplicated with ones for the other Endpoints.
(Mouse) Maximum packet size in bytes for Interrupt-In	0x8	Specify the maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
Interval for polling Interrupt-In EP for data transfers (milliseconds)	0x8	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt-In Endpoint. Value is in frame counts (1ms units for FS mode and 125us units for HS mode).
(Mouse) Interrupt-Out Endpoint (Optional)	Enable, Disable Default: Disable	This configuration is reserved and currently not used.
Endpoint Number for Interrupt-Out (Optional)	Endpoint 1-9 Default: Endpoint 4	This configuration is reserved and currently not used.
(Mouse) Maximum packet size in bytes for Interrupt-Out EP (Optional)	0x8	This configuration is reserved and currently not used.
(Mouse) Interval for polling Interrupt-Out EP for data transfers (milliseconds) (Optional)	0x8	This configuration is reserved and currently not used.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.

LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
User Callback for Device Event Notification (Only valid for USB Device)	NULL	Name must be a valid C symbol. The name of User defined USBX Device event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.

Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Device Class HID Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Device Class HID Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin

VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.42.6 Using the USBX Device Class HID Module in an Application

The configurator generates processing to create and register the USBX Device Class HID module; however, communication must be done after the device is connected to the host.

The typical steps in using the USBX Device Class HID module in an application are:

1. Get the ux_system_slave slave device pointer

2. Wait until slave device's ux_slave_device_state is configured

3. For HID event sending, use the ux_device_class_hid_event_set API

4. For received HID event reading, use the ux_device_class_hid_event_get API

5. For HID report sending, use the ux_device_class_hid_report_set API

6. For received HID report reading, use the ux_device_class_hid_report_get API

These common steps are illustrated in a typical operational flow diagram in the following figure:

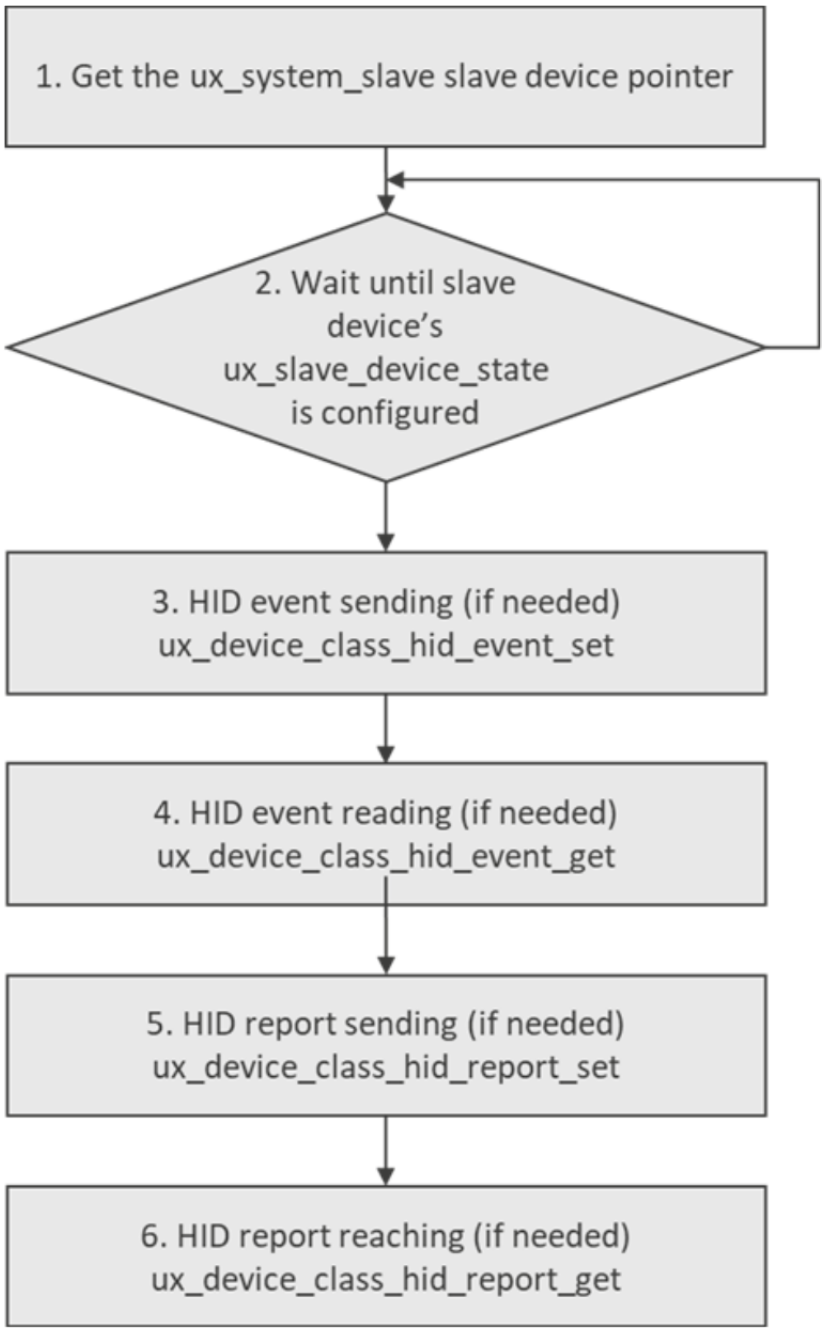


Figure 534: Flow Diagram of a Typical USBX Device Class HID Module Application

4.3.43 USBX Device Class Mass Storage

4.3.43.1 USBX Device Class Mass Storage Introduction

The USBX&trade Device Class Mass Storage module provides a high-level API for USB mass storage applications for USB Full Speed (USBFS) or USB High Speed (USBHS). The USBX Device Class Mass Storage module uses the USB and data-transfer peripherals on the Synergy MCU.

USBX Device Class Mass Storage Module Features

- ThreadX®-aware framework.
- Storage Media Parameter Setup
- Last LBA
- Byte-per-sector
- Type of storage media
- Removable flag
- USB Device Configuration (Device Configuration)
- Vendor ID
- Product ID
- Device Release Number
- Index of Serial Number String Descriptor
- Supported USB Specification (DCD)
- USBFS
- USBHS
- USB Device interrupts (DCD)

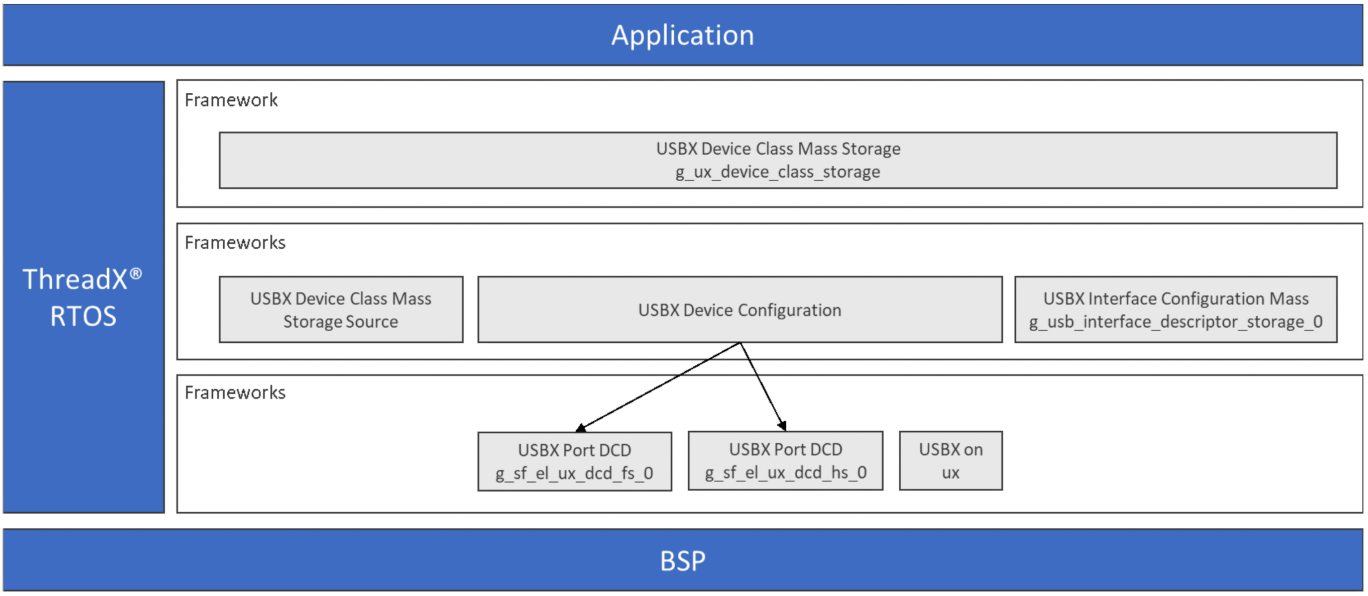


Figure 535: USBX Device Class Mass Storage Module Block Diagram

4.3.43.2 USBX Device Class Mass Storage Module APIs Overview

The USBX Device Class Mass Storage module automatically adds an initialization process; the user application only needs to prepare the callback functions for media access. Unless the functionality of the USBX Device Class Mass Storage module is required, there is no need to use it.

Note

For details on the USBX Device Stack, see the USBX Device Stack User's Manual.

4.3.43.3 USBX Device Class Mass Storage Module Operational Overview

The USBX Device Class Mass Storage module automatically adds an initialization process. The process initializes internal information with the given parameters and creates an internal thread for processing the mass-storage class; this internal thread processes all USB messages.

USBX Device Class Mass Storage Module Important Operational Notes and Limitations

USBX Device Class Mass Storage Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX Device Mass Storage (ux_device_class_storage)	6.1KB	19KB

Note

The information shown in the table above is valid if compiled with default USBX configurations.
The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 512(bytes) for S1 parts; 2048(bytes) for the other parts

- The USBX device storage class supports multiple logical unit numbers (LUNs), making it possible to create a storage device that acts simultaneously as a CD-ROM and flash disk.

USBX Device Class Mass Storage Module Limitations

- This module does not support the complex device.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.3.43.4 Including the USBX Device Class Mass Storage Module in an Application

This section describes how to include the USBX Device Class Mass Storage module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Device Class Mass Storage module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Device Class Mass Storage Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_device_class_storage USBX Device Class Mass Storage	Threads	New Stack> X-Ware> USBX> Device> Classes> Mass Storage> USBX Device Class Mass Storage

When the USBX Device Class Mass Storage module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

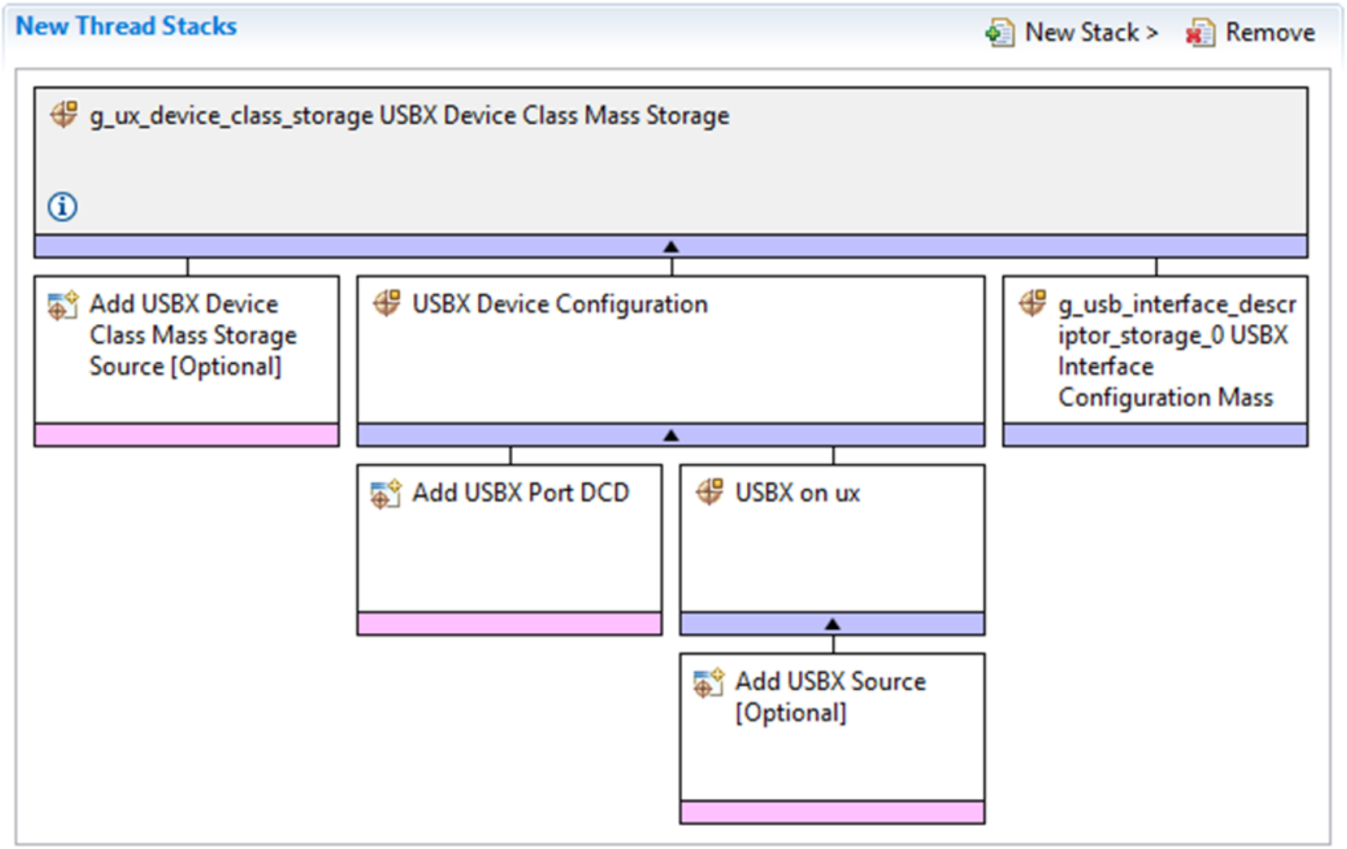


Figure 536: USBX Device Class Mass Storage Module Stack

4.3.43.5 Configuring the USBX Device Class Mass Storage Module

The USBX Device Class Mass Storage module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over

the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Device Class Mass Storage Module

ISDE Property	Value	Description
Name	g_ux_device_class_storage	Specify the name of USBX Interface Descriptor for Mass Storage Class. It must be a valid C symbol.
Mass Storage Class Parameter Setup	Auto (Simple Auto Setup if LUN is 1), Manual (User Manual Setup if LUN is greater than 1) Default: Auto	Manual (User Manual Setup if LUN is greater than 1). Simple Auto Setup if LUN is 1.
User Setup Callback (Only valid if Parameter Setup is Auto)	ux_device_class_storage_user_setup	Specify the name of user callback function to setup the storage parameter setup. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Last LBA of Storage Media (Only valid if Parameter Setup is Auto)	0	Specify the last LBA of storage media device (the number of sectors available in the media - 1). This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Bytes Per Sector of Storage Media (Only valid if Parameter Setup is Auto)	512	Specify the sector size of storage media. It can take multiple of 512 such as 512, 4K bytes. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Type of Storage Media (Only valid if Parameter Setup is Auto)	0	Specify the type of storage media device. Typically, the value takes following values. Flash Drive (0), CD-ROM device (5) This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".

Removable Flag of Storage Media (Only valid if Parameter Setup is Auto)	0x80	Specify the Removable Flag value of Storage Media. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Media Read Function Callback (Only valid if Parameter Setup is Auto)	ux_device_msc_media_read	Specify the C symbol name of Media Read callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when read access to the USB storage device is requested from user application. Refer USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device Storage Class" for the function definition.
Media Write Function Callback (Only valid if Parameter Setup is Auto)	ux_device_msc_media_write	Specify the C symbol name of Media Write callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when write access to the USB storage device is requested from user application. Refer USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device Storage Class" for the function definition.
Media Status Function Callback (Only valid if Parameter Setup is Auto)	ux_device_msc_media_status	Specify the C symbol name of Media Status callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when status inquiry to the USB storage device is requested from user application. Refer USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device Storage Class" for the function definition.

USBX Device Storage Instance Activate Callback Function	NULL	Specify the name of instance_activate user callback function for the USBX Device Mass Storage Class module. Name must be a valid C symbol. See the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device Storage Class" for more information about the instance_activate callback function.
USBX Device Storage Instance Deactivate Callback Function	NULL	Specify the name of instance_deactivate user callback function for the USBX Device Mass Storage Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual "Chapter 5: USBX Device Class Considerations USB Device Storage Class" for more information about the instance_activate callback function
Vendor ID Name	NULL	Specify the name of Vendor ID for the USBX Device Mass Storage Class module. Name must be a valid C symbol.
Product ID Name	NULL	Specify the name of Product ID for the USBX Device Mass Storage Class module. Name must be a valid C symbol.
Product Revision Number	NULL	Specify the number of Product revision for the USBX Device Mass Storage Class module. Value must be a non-negative integer.
Product Serial Number	NULL	Specify the number of Product serial for the USBX Device Mass Storage Class module. Value must be a non-negative integer.
Name of generated initialization function	ux_device_class_storage_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have

different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Device Class Mass Storage Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Device Class Mass Storage Source

ISDE Property	Value	Description
Maximum number of SCSI logical units	Value must be greater than 0 or empty Default: 2	UX_MAX_SLAVE_LUN This value represents the maximum number of SCSI logical units represented in the device storage class driver.
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Device Configuration Instance

ISDE Property	Value	Description
Vendor ID	0x045B	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (idVendor).
Product ID	0x0000	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (idProduct).
Device Release Number	0x0000	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (bcdDevice).

Index of Manufacturing String Descriptor	0x00	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iManufacturer). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.
Index of Product String Descriptor	0x00	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iProduct). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Index of Serial Number String Descriptor	0x00	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (iSerialNumber). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor specific Default: Communications(CDC)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (bDeviceClass).
Index of String Descriptor describing this configuration	0x00	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (iConfiguration). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.

Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (wTotalLength).
Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (bNumInterfaces).
Self-Powered	Enable, Disable Default: Enable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit6).
Remote Wakeup	Enable, Disable Default: Disable	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (bmAttributes bit5).
Maximum Power Consumption (in 2mA units)	50	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (bMaxPower).
Supported Language Code	0x0409	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.

Name of USBX String Framework	NULL	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Total index number of USB String Descriptors in USB String Framework	0	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USBX Language Framework	NULL	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.
Number of Languages to support (US English is applied if zero is set)	0	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.
Name of generated initialization function	ux_device_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Interface Configuration Mass Storage Instance

ISDE Property	Value	Description
Name	g_usb_interface_descriptor_storage_0	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.

Interface Number of Bulk Only Data Interface	0x00	Specify the index number of Bulk-Only Data interface. This configuration is a part of the USB Interface Descriptor (bInterface). The number must not be duplicated with any other Interface Numbers if your USB device consists of a USB composite device.
Endpoint Number to be used for Bulk Out Transfer	Endpoint 1-9 Default: Endpoint 1	Specify the Endpoint Number of Bulk Out Endpoint. It must not be duplicated with ones for the other Endpoints.
Endpoint Number to be used for Bulk In Transfer	Endpoint 1-9 Default: Endpoint 2	Specify the Endpoint Number of Bulk In Endpoint. It must not be duplicated with ones for the other Endpoints.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on USBHS

ISDE Property	Value	Description

High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Device Class Mass Storage Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Device Class Mass Storage Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description

Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.43.6 Using the USBX Device Class Mass Storage Module in an Application

- The USBX Device Class Mass Storage module does not need the usual initialization by an application; the application simply prepares the three user callbacks that the USBX Device Class Mass Storage module requires.
- Following is the suggested sequence of API's to be called successfully in a sequential order in an application for completely un-initializing USB Device class,

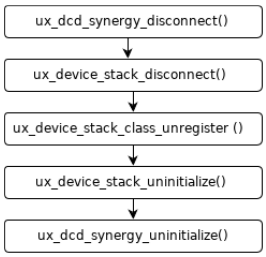


Figure 537: USBX Device Class Mass Storage Typical Application

4.3.44 USBX Host Class CDC-ACM

4.3.44.1 USBX Host Class CDC-ACM Module Introduction

The USBX Host Class CDC-ACM module provides a high-level API for USBX Host Class CDC-ACM applications and configures the USBX Host Class CDC-ACM Source, USBX Host Configuration, USBX Source, USBX Port HCD and a transfer driver. The USBX Host Class CDC-ACM module uses the DMAC/DTC and USB Host Class peripherals on the Synergy MCU.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

USBX Host Class CDC-ACM Module Features

The CDC-ACM class uses a composite device framework to group interfaces (control and data). As a result, care should be taken when defining the device descriptor. The USBX relies on the IAD descriptor to know internally how to bind interfaces. The IAD descriptor should be declared before the interfaces and contain the first interface of the CDC-ACM class and how many interfaces are attached. The CDC-ACM class also uses a union functional descriptor which performs the same function as the newer IAD descriptor. Although a union functional descriptor must be declared for historical reasons and compatibility with the host side, it is not used by the USBX.

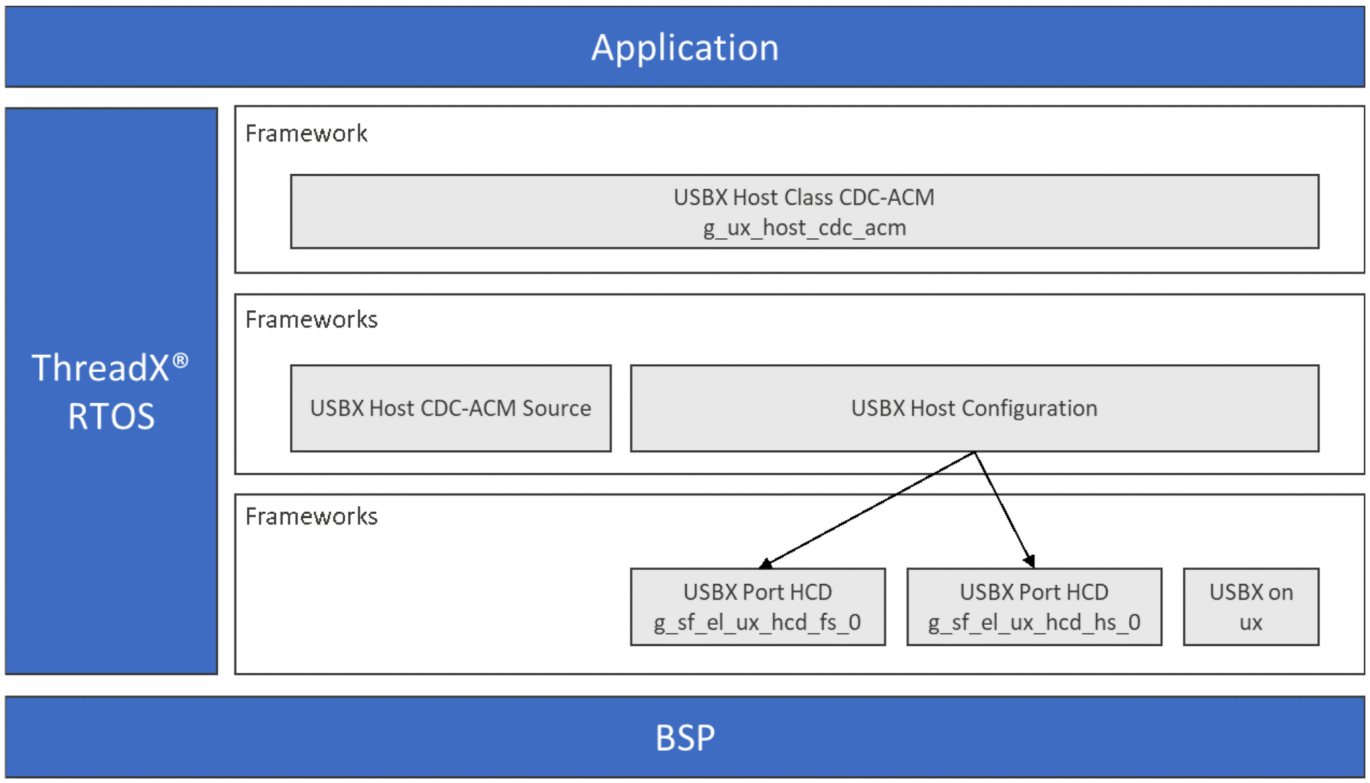


Figure 538: USBX Host Class CDC-ACM Module Block Diagram

4.3.44.2 USBX Host Class CDC-ACM Module APIs Overview

The USBX Host Class CDC-ACM Module defines APIs for reading, writing and ioctl. A complete list of

the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

USBX Host Class CDC-ACM Module Summary

Function Name	Example API Call and Description
ux_host_class_cdc_acm_read	status = ux_host_class_cdc_acm_read(cdc_acm, data_pointer, requested_length,&actual_length); This function reads from the cdc_acm interface. The call is blocking and only returns when there is either an error or when the transfer is complete.
ux_host_class_cdc_acm_write	status = ux_host_class_cdc_acm_write(cdc_acm, data_pointer,requested_length,&actual_length); This function writes to the cdc_acm interface. The call is blocking and only returns when there is either an error or when the transfer is complete.
ux_host_class_cdc_acm_ioctl	status = ux_host_class_cdc_acm_ioctl(cdc_acm, ioctl_function, &parameter_p); This function performs a specific ioctl function to the cdc_acm interface. The call is blocking and only returns when there is either an error or when the command is completed.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
UX_SUCCESS	The data transfer was completed.
UX_TRANSFER_TIMEOUT	Transfer timeout, reading/writing not completed.
UX_MEMORY_INSUFFICIENT	Not enough memory.
UX_HOST_CLASS_UNKNOWN	Wrong class instance.
UX_FUNCTION_NOT_SUPPORTED	Unknown IOCTL function.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.44.3 USBX Host Class CDC-ACM Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems

where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Host Class CDC-ACM Module Important Operational Notes and Limitations

USBX Host Class CDC-ACM Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX Host CDC-ACM (ux_host_class_cdc_acm)	N/A	30KB

Note

The information shown in the table above is valid if compiled with default USBX configurations. The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 512(bytes) for S1 parts; 2048(bytes) for the other parts

- Use a class container for the USBX Host Class CDC-ACM obtained by auto-generated code to get a CDC-ACM instance.
- Poll the flag ux_host_class_cdc_acm_state in the instance and make sure the status is live.
- Check if a DATA class interface is available in the CDC-ACM instance.
- Set up the CDC-ACM reception if required.
- Perform CDC-ACM communication with USB device.

USBX Host Class CDC-ACM Module Limitations

- The module needs the interrupt of a USB Controller to be enabled.
- See SSP Release Notes (sf_el_ux sections) for known limitations.
- The module uses the interrupt of a USB Controller. Set the appropriate interrupt priority level in Synergy Configuration tool. By default, it is disabled.
- The module uses the interrupt of a Transfer module (implemented as DMAC or DTC) if it is used. Set appropriate priority level in the Synergy Configuration tool. The level must be higher than a USB Controller to work properly.
- A zero length packet does not contain the received data, but the application must perform the receive operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.44.4 Including the USBX Host Class CDC-ACM Module in an Application

This section describes how to include the USBX Host Class CDC-ACM Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class CDC-ACM Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Host Class CDC-ACM Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_device_class_cdc_acm0 USBX Host Class CDC-ACM	Threads	New Stack> X-Ware> USBX> Device> Classes > CDC-ACM > USBX Host Class CDC-ACM

When the USBX Host Class CDC-ACM Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

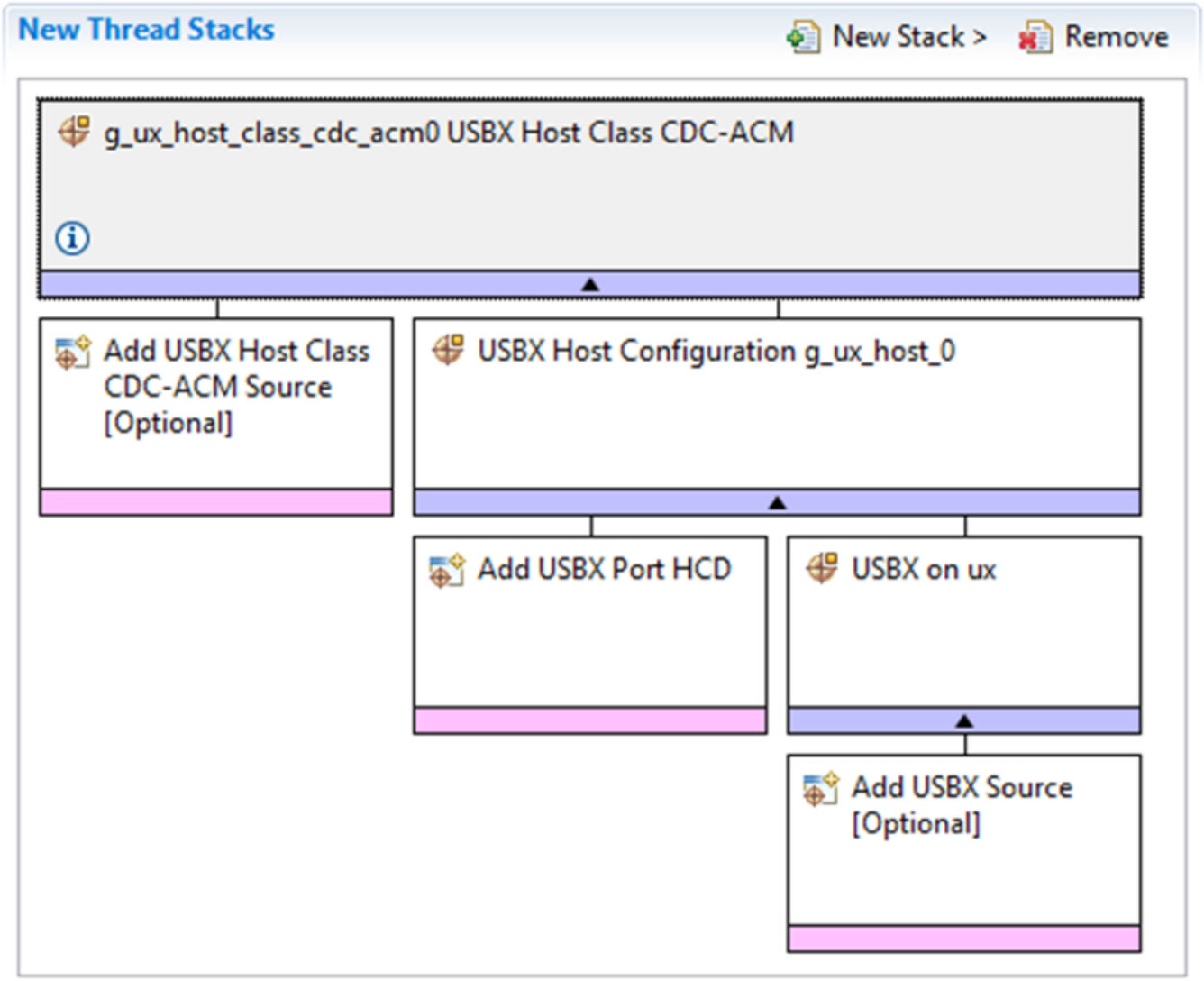


Figure 539: USBX Host Class CDC-ACM Module Stack

4.3.44.5 Configuring the USBX Host Class CDC-ACM Module

The USBX Host Class CDC-ACM Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class CDC-ACM Module

ISDE Property	Value	Description
Name	g_ux_host_class_cdc_acm0	Specify the name of USBX Host CDC-ACM Class module instance. It must be a valid C symbol.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class CDC-ACM Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Host Class CDC-ACM Source

ISDE Property	Value	Description
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Host Configuration Instance

ISDE Property	Value	Description
Name	g_ux_host0	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.
Name of generated initialization function	ux_host_hid_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description

High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0,1,2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Host Class CDC-ACM Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class CDC-ACM Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.44.6 Using the USBX Host Class CDC-ACM Module in an Application

The configurator generates processing to create and register the USBX Host Class CDC-ACM module; however, communication must be done after the Host is connected to the host.

The typical steps in using the USBX Host Class CDC-ACM module in an application are:

1. Get the first instance of the connected device with ux_host_stack_class_instance_get API.

2. Wait for the device status to become live.

3. Check that the class of the device is CDC Data Class.

4. If there is the next device, check the status again.

5. For received data reading, use the ux_host_class_cdc_acm_read API.

6. For data sending, use the ux_host_class_cdc_acm_write API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

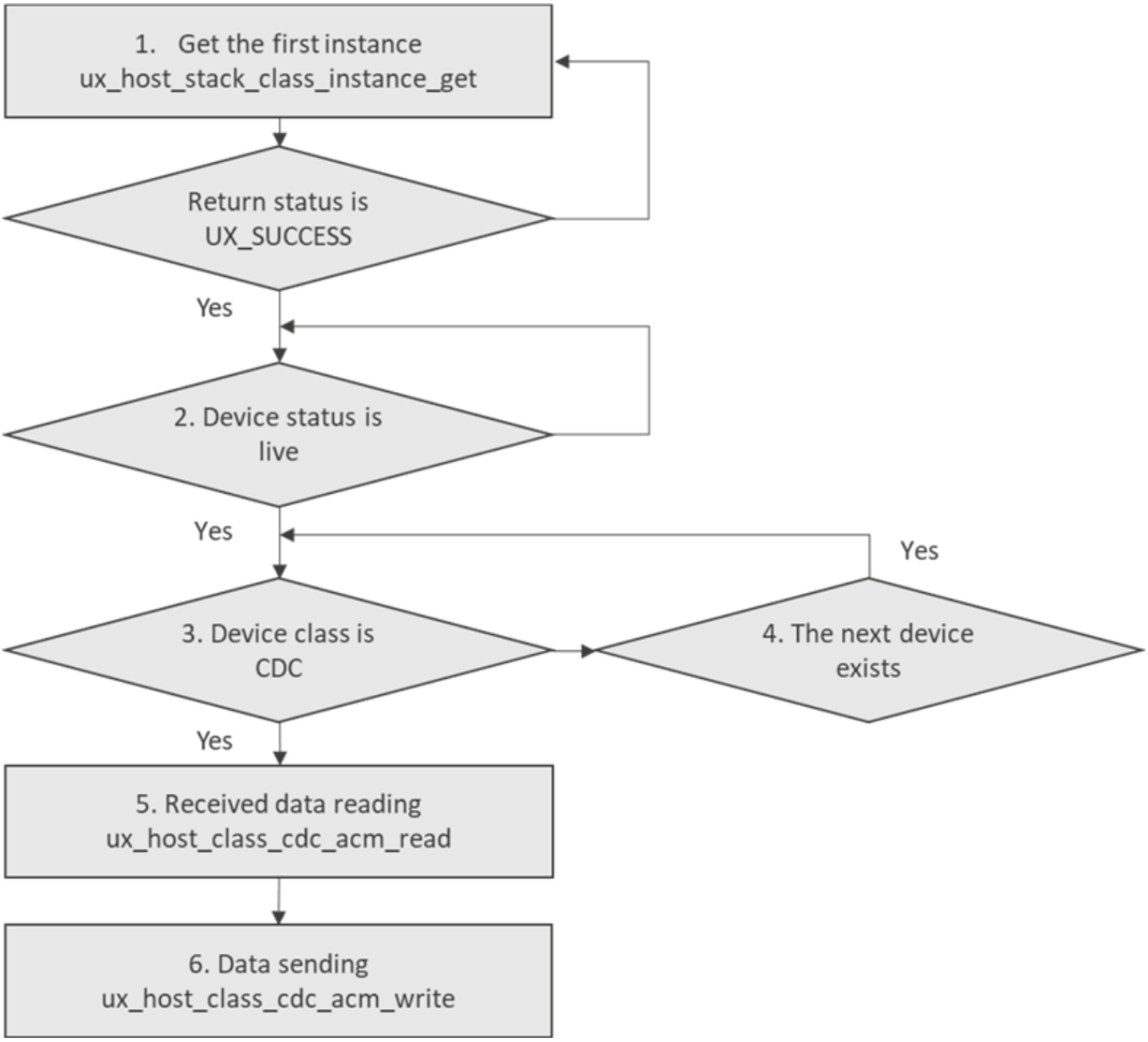


Figure 540: Flow Diagram of a Typical USBX Host Class CDC-ACM Module Application

4.3.45 USBX Host Class HID

4.3.45.1 USBX Host Class HID Module Introduction

The USBX™ Host Class HID module provides a high-level API for human interface device (HID) applications and configures the USBX Host Class HID Source, USBX Host Configuration, USBX Source and USBX Port HCD. The USBX Host Class HID module uses the USB peripheral on the Synergy MCU.

USBX Host Class HID Module Features

The USBX Host Class Human Interface Device (HID) module supports the USBX HID class. It provides the following features:

- Supports HID report data transfers
- Supports the following clients:
  - Keyboard
  - Mouse
  - Remote-Control

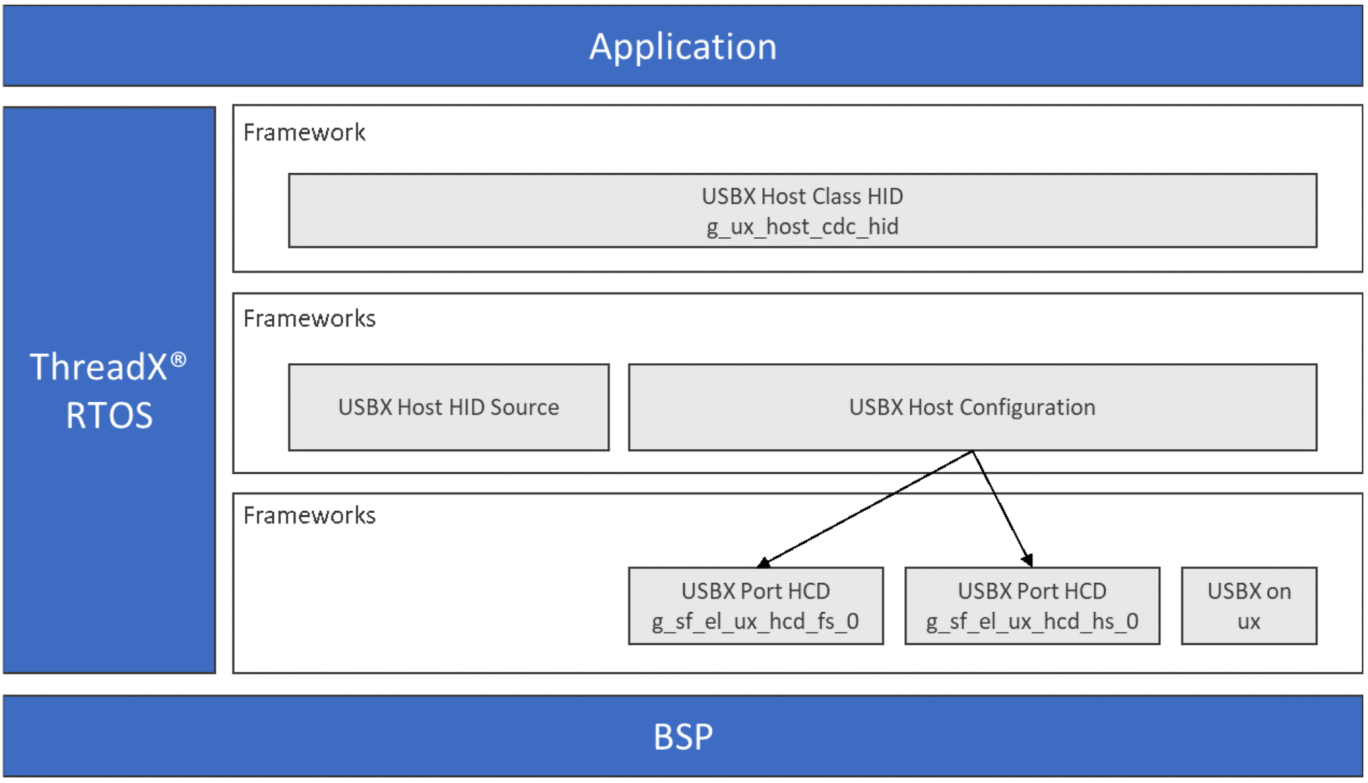


Figure 541: USBX Host Class HID Module Block Diagram

4.3.45.2 USBX Host Class HID Module APIs Overview

The USBX Host Class HID module defines APIs for registering callbacks, starting and stopping periodic reports and reporting gets and sets. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

USBX Host Class HID Module Summary

Function Name	Example API Call and Description

ux_host_class_hid_report_callback_register	ux_host_class_hid_report_callback_register(hid, &call_back); This function is used to register a callback from the HID class to the HID client when a report is received.
ux_host_class_hid_periodic_report_start	ux_host_class_hid_periodic_report_start(hid); This function is used to start the periodic (interrupt) endpoint for the instance of the HID class that is bound to this HID client.
ux_host_class_hid_periodic_report_stop	ux_host_class_hid_periodic_report_stop(hid); This function is used to stop the periodic (interrupt) endpoint for the instance of the HID class that is bound to this HID client.
ux_host_class_hid_report_get	ux_host_class_hid_report_get(hid, &client_report); This function is used to receive a report directly from the device without relying on the periodic endpoint.
ux_host_class_hid_report_set	ux_host_class_hid_report_set(hid, &client_report); This function is used to send a report directly to the device.
ux_host_class_hid_keyboard_key_get	ux_host_class_hid_keyboard_key_get(keyboard_instance, &keyboard_char, &keyboard_state); This function is used to read the keyboard data received from the device.
ux_host_class_hid_mouse_buttons_get	ux_host_class_hid_mouse_buttons_get(&mouse_instance, &mouse_buttons); This function is used to read the mouse button information received from the device.
ux_host_class_hid_mouse_position_get	ux_host_class_hid_mouse_position_get(mouse_instance, &x_position, &y_position); This function is used to read the position information of the mouse received from the device.
ux_host_class_hid_remote_control_usage_get	ux_host_class_hid_remote_control_usage_get(remote_control_instance, &usage, &value); This function is used to read the remote controller information received from the device.

Note

For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description

UX_SUCCESS	The data transfer was completed.
UX_TRANSFER_TIMEOUT	Transfer timeout, reading/writing not completed.
UX_MEMORY_INSUFFICIENT	Not enough memory.
UX_HOST_CLASS_UNKNOWN	Wrong class instance.
UX_FUNCTION_NOT_SUPPORTED	Unknown IOCTL function.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.45.3 USBX Host Class HID Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Host Class HID Module Important Operational Notes and Limitations

USBX Host Class HID Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX Host HID (ux_host_class_hid)	N/A	HID Mouse: 38KB HID Keyboard: 46KB

The memory size of the USBX Host HID as shown in the table above is with the following configurations (default settings) being made. The size can be reduced if the Source module is used.

- UX_HOST_CLASS_HID_DECOMPRESSION_BUFFER: 4096(bytes)
- UX_HOST_CLASS_HID_USAGE: 1024(WORDs)

Note

The information shown in the table above is valid if compiled with default USBX configurations.
The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 512(bytes) for S1 parts; 2048(bytes) for the other parts

- By default, the module supports keyboard, mouse, and remote control clients.
- Use a class container for the USBX Host Class HID that is obtained by auto-generated code to get an HID instance.
- Pole the flag, ux_host_class_hid_state, in the instance and ensure the status is live.
- Check whether or not a client local instance is available in the HID instance.
- Perform HID communication with the USB device.

USBX Host Class HID Module Limitations

- The module uses the interrupt of the USB Controller, which needs to be enabled. For proper operation, set the appropriate interrupt-priority level in the Synergy Configuration tool.
- If transfer module is used, the module uses the interrupt (implemented as DMAC or DTC). Set the priority level in the Synergy Configuration tool. The level has to be higher than a USB Controller; otherwise, it does not work.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.45.4 Including the USBX Host Class HID Module in an Application

This section describes how to include the USBX Host Class HID Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class HID Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Host Class HID Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_host_class_hid0 USBX Host Class HID	Threads	New Stack> X-Ware> USBX> Host > Classes > HID > USBX Host Class HID

When the USBX Host Class HID Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

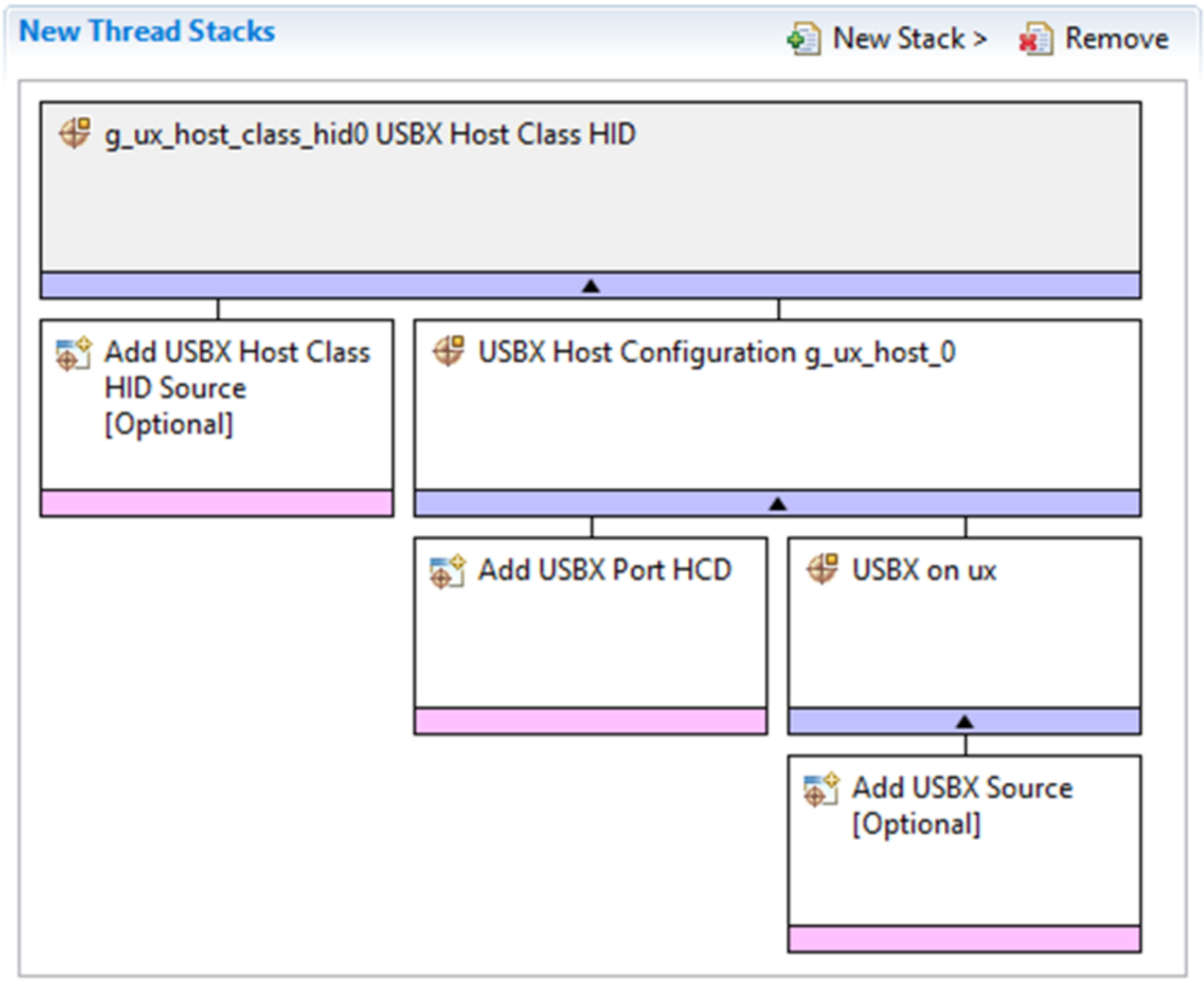


Figure 542: USBX Host Class HID Module Stack

4.3.45.5 Configuring the USBX Host Class HID Module

The USBX Host Class HID Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class HID Module

ISDE Property	Value	Description
Name	g_ux_host_class_hid0	Specify the name of USBX Host Mass HID module instance. It must be a valid C symbol.
HID Client - Keyboard Support	Enable, Disable Default: Enable	Set Enable to support USB keyboard devices. This configuration registers the USBX HID Keyboard Client.
HID Client - Mouse Support	Enable, Disable Default: Enable	Set Enable to support USB mouse devices. This configuration registers the USBX HID Mouse Client.
HID Client - Remote Control Support	Enable, Disable Default: Enable	Set Enable to support USB remote control devices. This configuration registers the USBX Remote Control Client.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class HID Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Host Class HID Source

ISDE Property	Value	Description
HID Keyboard Thread Priority	Value must be greater than 0 or empty Default: 20	UX_THREAD_PRIORITY_KEYBOARD Define the priority of the HID keyboard thread.
Memory size for HID Report Decompression	Value must be greater than 0 or empty Default: 4096	UX_HOST_CLASS_HID_DECOMPRESSION_BUFFER Define the memory size to build a decompressed report
Number of Entries for HID Local Usage Item Table	Value must be greater than 0 or empty Default: 1024	UX_HOST_CLASS_HID_USAGES Define the size of HID local usage item table. One item entry consumes 4 bytes.

Host HID Interrupt Out Support	Enabled, Disabled Default: Disabled	UX_HOST_CLASS_HID_INTERRUPT_OUT_SUPPORT When enabled, host HID interrupt OUT transfer is supported.
HID report transfer timeout	Value must be greater than 0 or empty Default: 10000	UX_HOST_CLASS_HID_REPORT_TRANSFER_TIMEOUT When set, this represents the HID report transfer timeout value in millisecond.
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Host Configuration Instance

ISDE Property	Value	Description
Name	g_ux_host0	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.
Name of generated initialization function	ux_host_hid_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.

VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0,1,2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.

USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Host Class HID Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class HID Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description

Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin

PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.45.6 Using the USBX Host Class HID Module in an Application

The configurator generates processing to create and register the USBX Host Class HID Module; however, communication must be done after the Host is connected to the host.

The typical steps in using the USBX Host Class HID Module in an application are:

1. Get the first instance of the connected device with the ux_host_stack_class_instance_get API.

2. Wait for ux_success.

3. Wait for device status live.

4. Wait for client instance live.

5. If the Device is a keyboard.

6. Receive keyboard data using the ux_host_class_hid_keyboard_key_get API.

7. If the Device is a mouse.

8. Receive mouse data using the ux_host_class_hid_mouse_buttons_get API and ux_host_class_hid_mouse_position_get API.

9. If the Device is a remote controller.

10. Receive remote controller data using the ux_host_class_hid_remote_control_usage_get API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

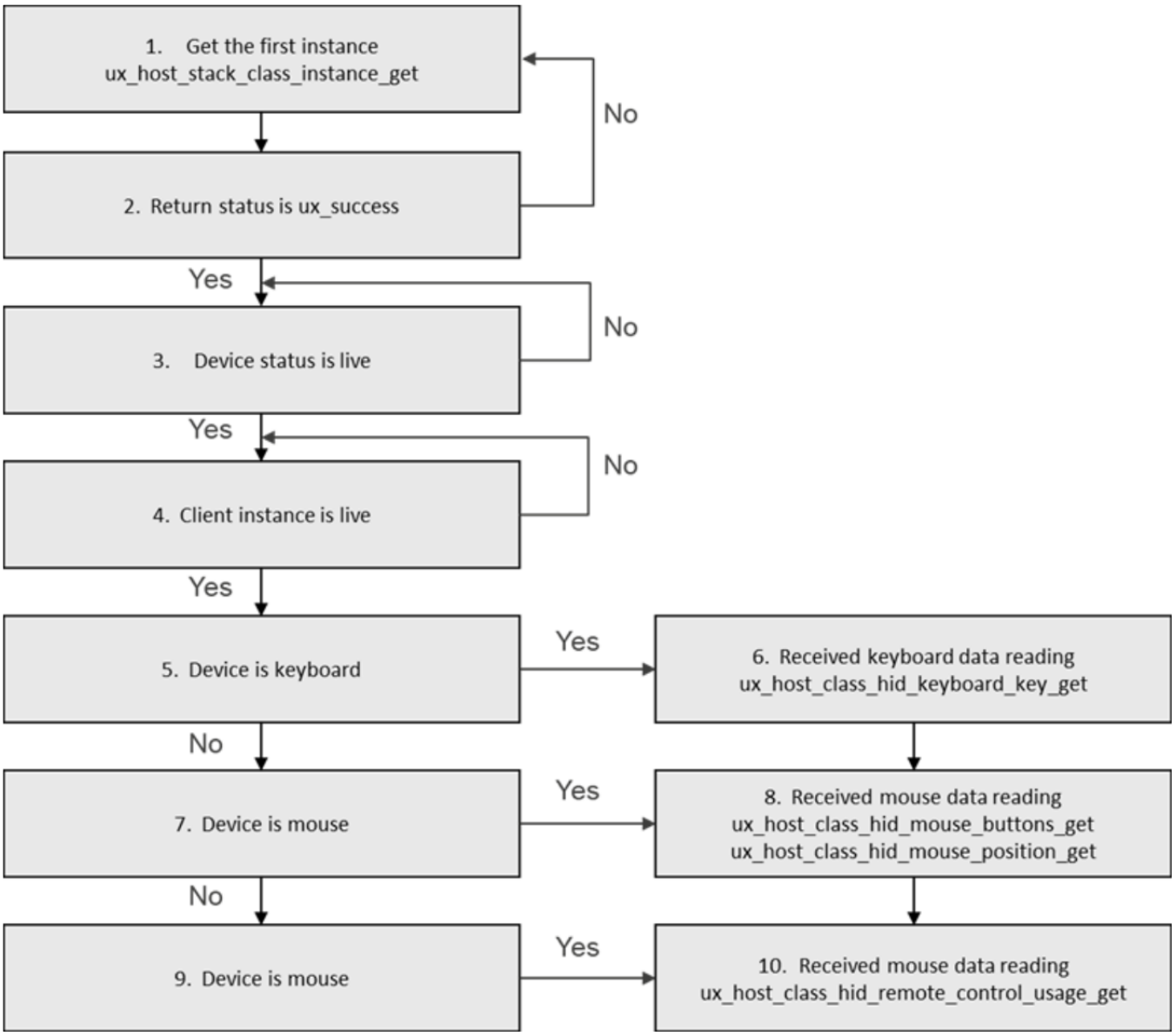


Figure 543: Flow Diagram of a Typical USBX Host Class HID Module Application

4.3.46 USBX Host Class HUB

4.3.46.1 USBX Host Class Hub Module Introduction

The USBX™ Host Class Hub module provides a high-level API for USBX Host Class Hub applications and configures the USBX Host Class Hub Source, USBX Host Configuration, USBX Source and USBX Port Host Controller Device. The USBX Host Class Hub module uses the USB peripheral on the Synergy MCU.

USBX Host Class Hub Module Features

- Supports either a stand-alone hub or functions as part of a compound device such as a keyboard or a monitor.
- Supports either self-powered or bus-powered modes.
- Bus-powered hubs have a maximum of four downstream ports.
- Hubs can be cascaded.
- Up to five hubs can be connected to one another.
- Supports the connection of devices that are either self-powered or bus-powered using less than 100 mA of power.

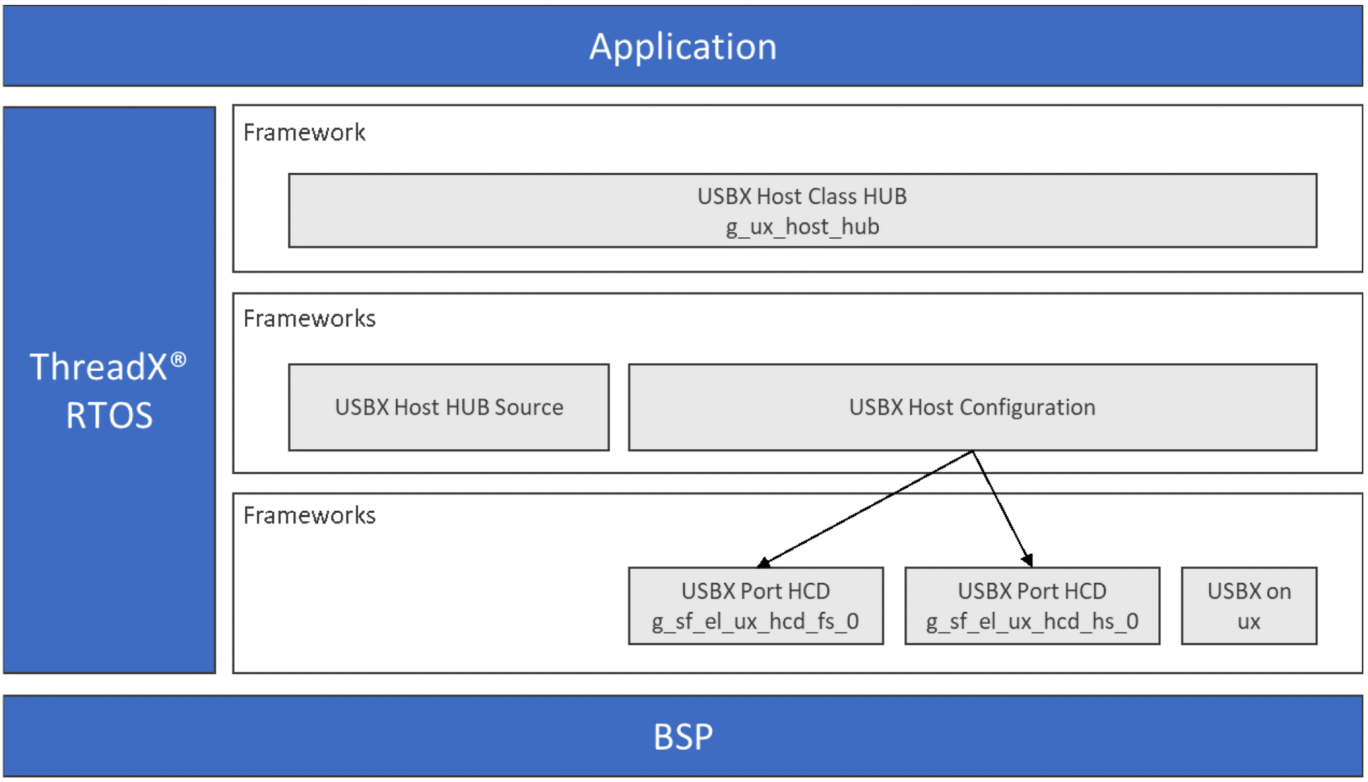


Figure 544: USBX Host Class Hub Module Block Diagram

4.3.46.2 USBX Host Class Hub Module APIs Overview

The USBX Host Class Hub Module does not have an API for the user application.

4.3.46.3 USBX Host Class Hub Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Host Class Hub Module Important Operational Notes and Limitations

USBX Host Class Hub Module Operational Notes

The Hub class is registered by the auto-generated code. In the user application, no special operation other than the registration of the Hub class is necessary.

USBX Host Class Hub Module Limitations

- The module needs the interrupt of a USB Controller enabled.
- The module uses the interrupt of a USB Controller. Set appropriate interrupt-priority level in the Synergy Configuration tool for proper operation.
- The module uses the interrupt of a transfer module (implemented as DMAC or DTC) if one is implemented. Set the appropriate priority level in the Synergy Configuration tool. The priority level must be higher than that of the USB Controller for proper operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.46.4 Including the USBX Host Class Hub Module in an Application

This section describes how to include the USBX Host Class Hub Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class Hub Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Host Class Hub Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_host_class_hub0 USBX Host Class Hub	Threads	New Stack> X-Ware> USBX> Host > Classes > Hub > USBX Host Class Hub

When the USBX Host Class Hub Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

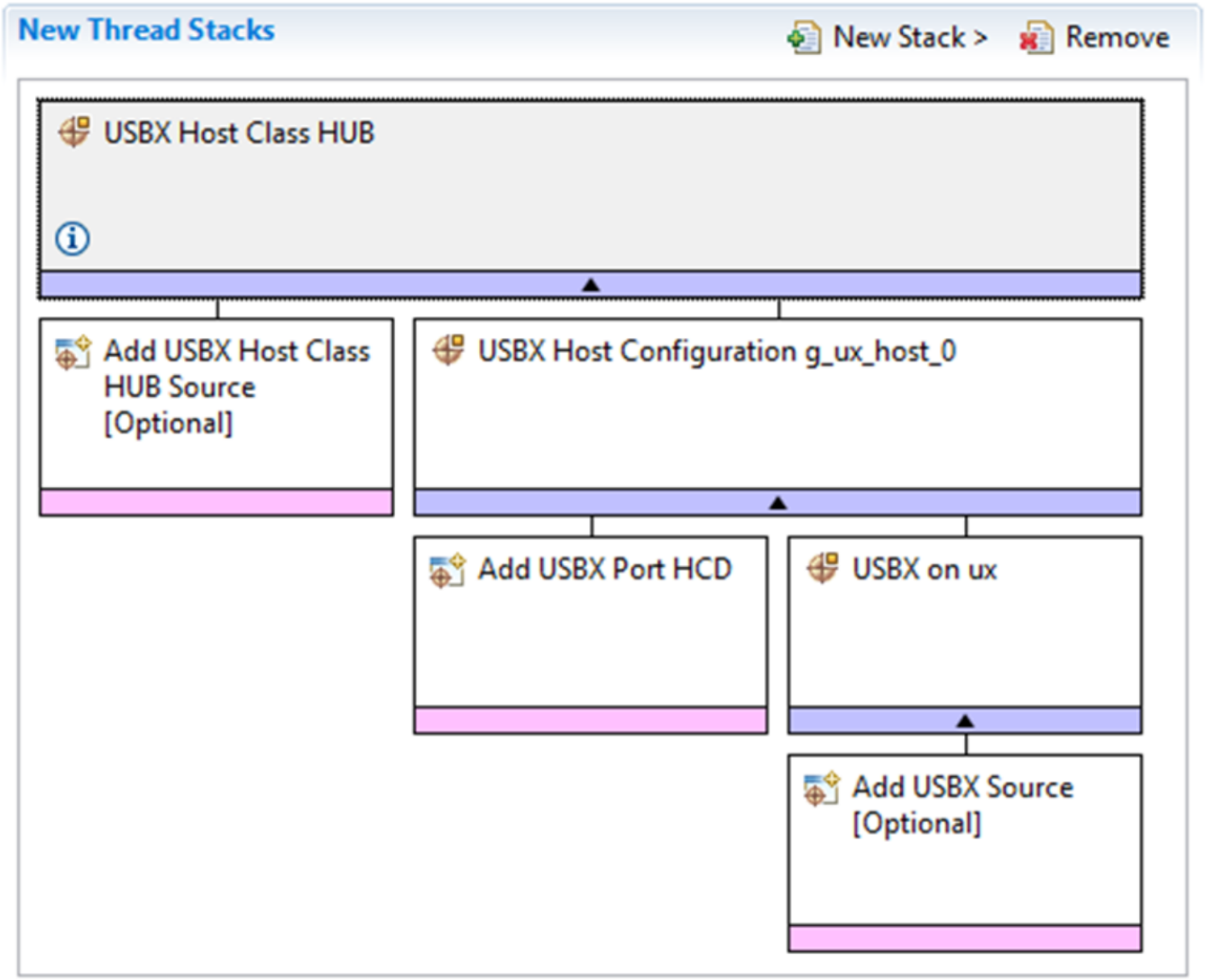


Figure 545: USBX Host Class Hub Module Stack

4.3.46.5 Configuring the USBX Host Class Hub Module

The USBX Host Class Hub Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class Hub Module

ISDE Property	Value	Description
No configurable properties		

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class Hub Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Host Class Hub Source

ISDE Property	Value	Description
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Host Configuration Instance

ISDE Property	Value	Description
Name	g_ux_host0	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.
Name of generated initialization function	ux_host_hid_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0, 1, 2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have

different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Host Class Hub Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class Hub Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin

ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.46.6 Using the USBX Host Class Hub Module in an Application

An application typically does not use the Hub module by itself; other classes (CDC-ACM, Storage, HID and so on.) are also used at the same time.

The configurator generates processing to register the USBX Host Class Hub module. Specify the same module as USBX Host Configuration module registered in the class to be used at the same time.

The following figure shows the stack when registered with the USBX Host Class Mass Storage module at the same time:

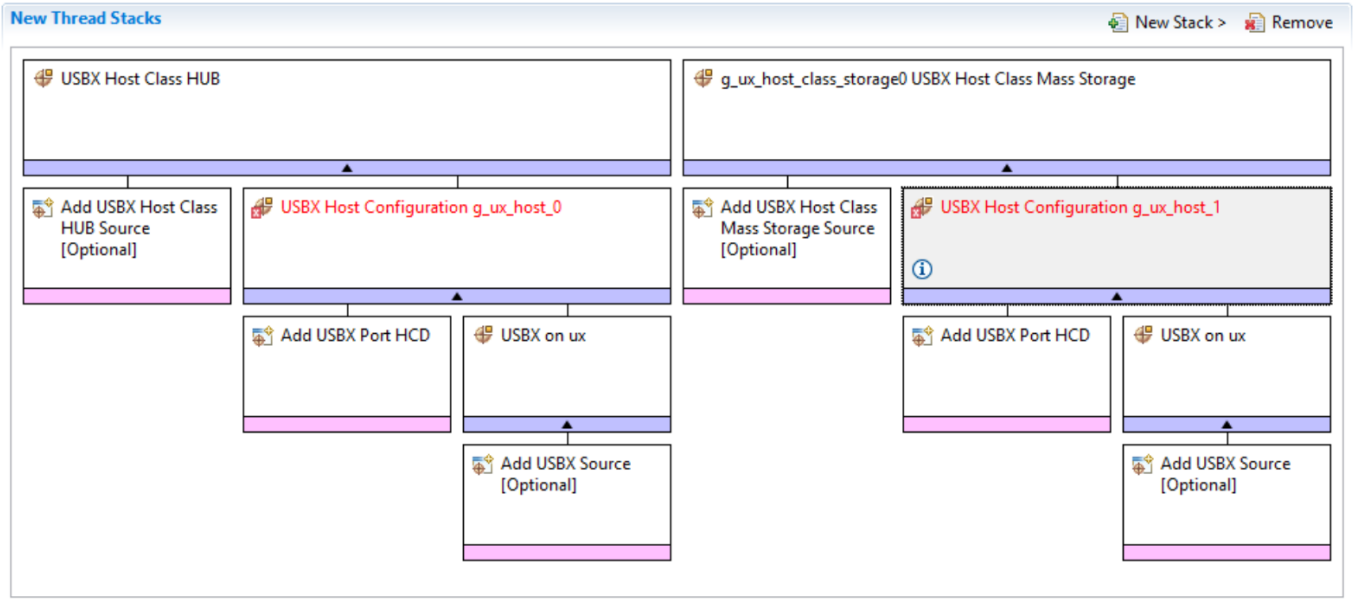


Figure 546: Flow Diagram of a Typical USBX Host Class Hub Module Application

4.3.47 USBX Host Class Printer

4.3.47.1 USBX Host Class Printer Module Introduction

The USBX™ Host Class Printer module provides a high-level API for USBX Host Class Printer module applications and uses the USB and data-transfer peripherals on the Synergy MCU. A user defined callback can be created to determine when the stack activates or deactivates the USB Printer class.

Unsupported Features

USBX host class PIMA is not supported in this version of SSP.

USBX Pictbridge is not supported in this version of SSP.

USBX Host Class Printer Module Features

The USB Host Class Printer module allows for a USB host-system to communicate with the Printer. This class is based on the USB standard. The USBX Host Class Printer module includes the following key features:

- Support for USB Full Speed (USBFS) or USB High Speed (USBHS)
- Receive and transmit data-transfer drivers for improved performance
- High-level APIs for reading and writing

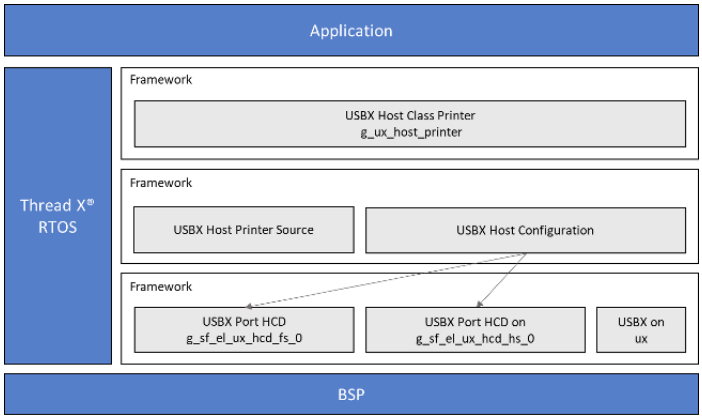


Figure 547: USBX Host Class Printer Module Block Diagram

4.3.47.2 USBX Host Class Printer Module APIs Overview

The USBX Host Class Printer Module defines APIs, which are used to interact with the Printer interface. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

USBX Host Class PRINTER Module Summary

Function Name	Example API Call and Description

ux_host_class_printer_read	UINT ux_host_class_printer_read (UX_HOST_CLASS_PRINTER *printer, UCHAR *data_pointer, ULONG requested_length, ULONG *actual_length) This function reads from the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete. A read is allowed only on bi-directional printers.
ux_host_class_printer_write	UINT ux_host_class_printer_write (UX_HOST_CLASS_PRINTER *printer,UCHAR *data_pointer, ULONG requested_length,ULONG *actual_length) This function writes to the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete.
ux_host_class_printer_soft_reset	UINT ux_host_class_printer_soft_reset (UX_HOST_CLASS_PRINTER *printer) This function performs a soft reset to the printer
ux_host_class_printer_status_get	UINT ux_host_class_printer_status_get (UX_HOST_CLASS_PRINTER *printer,ULONG *printer_status) This function obtains the printer status. The printer status is similar to the LPT status (1284 standard).
_ux_host_class_printer_name_get	UINT _ux_host_class_printer_name_get(UX_HOST_CLASS_PRINTER *printer) This function obtains the Device ID string.

Note

for more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Return Values

Name	Description
UX_SUCCESS	This operation was successful.
UX_FUNCTION_NOT_SUPPORTED	Function not supported because the printer is not bi-directional.
UX_TRANSFER_TIMEOUT	Transfer timeout, reading or writing is incomplete, or reset not completed.
UX_MEMORY_INSUFFICIENT	Not enough memory to perform the operation.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.47.3 USBX Host Class Printer Module Operational Overview

The initialization of the Printer class expects some specific parameters as illustrated in the application project associated with this module guide.

Printers have two different types of commands: those that transfer data and those that control the USB interface or Printer interface. The host prints something on a printer by delivering data on the Bulk OUT endpoint. This data is in the form of PostScript, HP PCL or any other PDL. This data may also be encapsulated in a PCP, such as IEEE 1284.1, or something that is vendor-specific. In addition, the data may also be simple text, or it may be a proprietary PDL. User application should take care of converting data into printer supported formats. The printer can respond periodically on the Bulk IN endpoint with status regarding the data it is receiving, or because of an asynchronous event. A typical printed page takes the following sequence:

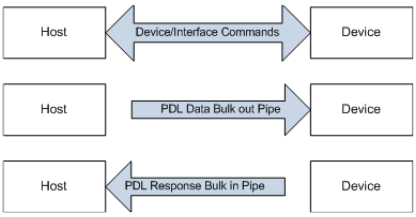


Figure 548: Printing a page

The PDL data is sent to the device on the Bulk OUT pipe. If the device uses a PCP, then the PDL is encapsulated in the PCP; the Bulk IN pipe is used for any responses, such as errors and printer status as defined in the PCP. For a unidirectional interface, the status is retrieved by issuing a port status command on the default pipe, and the status is returned on the default pipe.

USBX Host Class Printer Module Important Operational Notes and Limitations

USBX Host Class Printer Module Operational Notes

The USBX Host stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

USBX Class	S1 Parts	Other Parts
USBX HostPrinter(ux_Host_class_printer)	N/A	27KB

Note

the information shown in the table above is valid if compiled with default USBX configurations.
the memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:

UX_THREAD_STACK_SIZE: 2048 (bytes) for the other parts

USBX Host Class Printer Module Limitations

- Refer to the most recent *SSP Release Notes* for any additional operational limitations for

this module.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.47.4 Including the USBX Host Class Printer Module in an Application

This section describes how to include the USBX Host Class Printer Module in an application using the SSP configurator.

Note

it is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class Printer Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBXHost Class Printer Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_Host_class_printer0 USBX Host Class Printer	Threads	New Stack> X-Ware> USBX>Host> Classes >Printer> USBX Host Class Printer

When the USBX Host Class Printer Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes Add in the text. Clicking on any Pink banded modules brings up the new icon and displays possible choices.

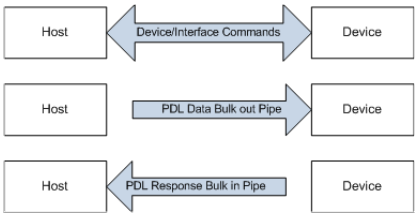


Figure 549: USBX Host Class Printer Module Stack

4.3.47.5 Configuring the USBX Host Class Printer Module

The USBX Host Class Printer Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than manual

approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class Printer Module

ISDE Property	Value	Description
Name	g_ux_Host_class_printer0	Specify the name of the USBX Host Printer Class module instance. It must be a valid C symbol.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.
Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class Printer Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Host Class Printer Source

ISDE Property	Value	Description
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn user's possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

the example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Host Configuration Instance

ISDE Property	Value	Description
Name	g_ux_host0	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.

Name of generated initialization function	ux_host_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

the example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

the example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0,1,2 Default: 0	Select the number of isochronous pipes to reserve.

VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.

Note

the example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size		See section "Azure RTOS USBX Memory Requirements" for the required memory size for each class.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

the example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Host Class Printer Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class Printer Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external Hosts. Select I/O pins and configure to the external Host requirements. The following table lists the pin selection method within the SSP Configuration Window, and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

the selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

the selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Host, Host, OTG Default: Custom	Select Host as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

the example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description

Operation Mode	Disabled, Custom, Host, Host, OTG Default: Custom	Select Host as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

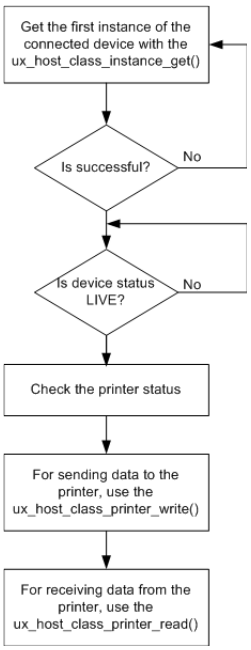
the example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.47.6 Using the USBX Host Class Printer Module in an Application

The configurator generates processing to create and register the USBX Host Class CDC-ACM module; however, communication must be done after the Printer is connected to the host.

The typical steps in using the USBX Host Class Printer module in an application are:

1. Get the first instance of the connected device with the ux_host_class_instance_get API.
2. Wait until successful.
3. Wait for the device status to become live.
4. Check the status of the printer.
5. For data sending to the printer, use the ux_host_class_printer_write API.
6. For data receiving from printer, use the ux_host_class_printer_read API.



4.3.48 USBX Host Class Mass Storage

4.3.48.1 USBX Host Class Mass Storage Module Introduction

The USBX™ Host Class Mass Storage module provides a high-level API for USBX Host Class Mass Storage applications and uses the USB and data-transfer peripherals on the Synergy MCU.

USBX Host Class Mass Storage Module Features

- Host controller for USB 2.0 with support for:
  - Root Hub
  - Power Management
  - Endpoints
  - Transfers
- High-level APIs simplify storage operations
- Supports optional data transfers using MCU hardware for improved efficiency

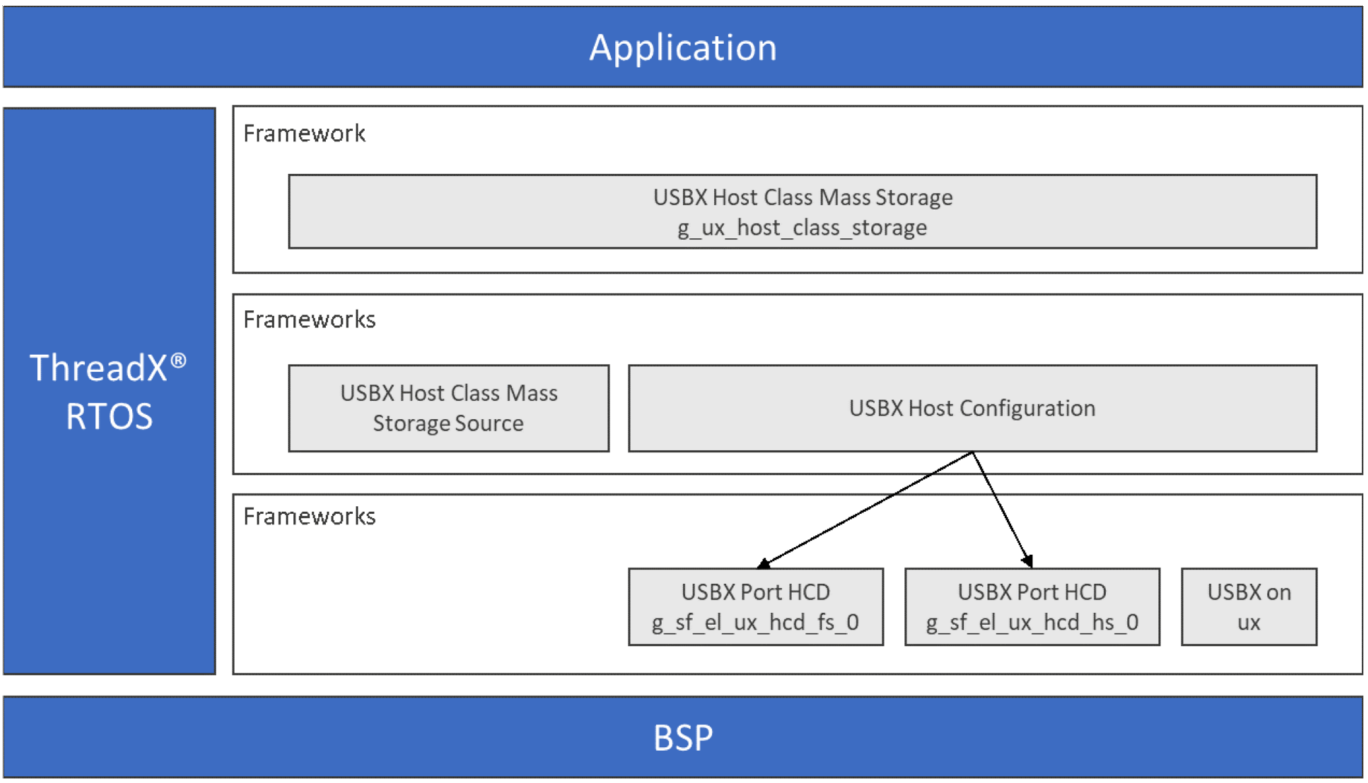


Figure 550: USBX Host Class Mass Storage Module Block Diagram

4.3.48.2 USBX Host Class Mass Storage Module APIs Overview

The USBX Host Class Mass Storage module does not have a separate API available for user applications. When a USB media is connected, it is attached to the FileX® member included in the USBX Host Class Mass Storage instance. The user application uses this FileX member to access files on the USB media. For FileX APIs, refer to the FileX User Guide for the Renesas Synergy™ Platform for more details.

4.3.48.3 USBX Host Class Mass Storage Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Host Class Mass Storage Module Important Operational Notes and Limitations

USBX Host Class Mass Storage Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

__Memory (RAM) Requirements for the USBX Memory Pool__

USBX Class	S1 Parts	Other Parts
USBX Host Mass Storage (ux_host_class_storage)	N/A	Pre-built library: 42KB Source: 39KB + UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE + UX_HOST_CLASS_STORAGE_THREAD_STACK_SIZE+ (Add the additional memory as per Note*1)

Note

**1: If 'Maximum TDs' is exceeds more then 128 in (USBX Source module: USBX Source property >> Common >> Maximum TDs) in the XML configurator, add the additional memory to the USBX memory pool as follows:*

- *Note*

  *(Maximum TDs - 128)*48 = Additional memory in Bytes.*
  The information shown in the table above is valid if compiled with default USBX configurations.
  The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds: UX_THREAD_STACK_SIZE: 512 (bytes) for S1 parts; 2048 (bytes) for the other parts
  The memory size of the USBX Host Mass Storage is also shown with the case built with the Source module because the size depends on the configuration UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE and would differ much from the pre-build library's. Note that the size is with following configuration applied: UX_THREAD_STACK_SIZE: 2048 (bytes)

USBX Host Class Mass Storage Module Operational Procedure

- Use a class container for the USBX Host Class Mass Storage module obtained by auto-generated code to get a mass storage instance.
- Poll the flag ux_host_class_storage_state in and make sure the status is live.
- Wait until the first media of the class container is attached.
- Access the file on the media with the FileX API.

USBX Host Class Mass Storage Module Limitations

- The module uses the interrupt of a USB controller; set the appropriate interrupt-priority level in the Synergy Configuration tool to ensure proper operation.
- The module uses the interrupt of a transfer module (implemented as DMAC or DTC); set the appropriate priority level in the Synergy Configuration tool. The level must be higher than a USB Controller to ensure proper operation.
- USBX Host Class Mass storage module only supports the MBR partition type on the storage device.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.48.4 Including the USBX Host Class Mass Storage Module in an Application

This section describes how to include the USBX Host Class Mass Storage Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class Mass Storage Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Host Class Mass Storage Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence

g_ux_host_class_storage USBX Host Class Mass Storage	Threads	New Stack> X-Ware> USBX> Device> Classes> Mass Storage > USBX Host Class Mass Storage

When the USBX Host Class Mass Storage Module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

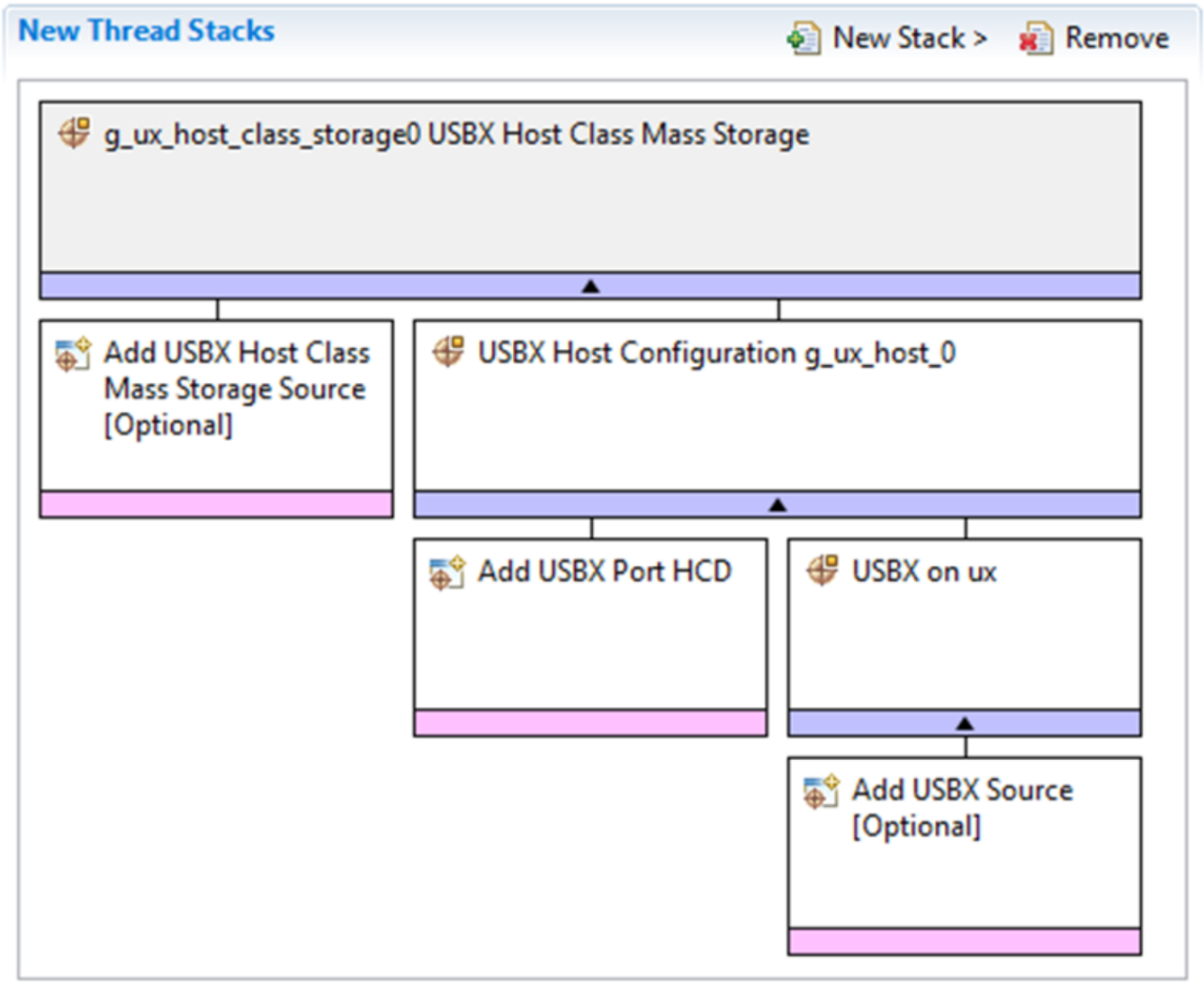


Figure 551: USBX Host Class Mass Storage Module Stack

4.3.48.5 Configuring the USBX Host Class Mass Storage Module

The USBX Host Class Mass Storage Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class Mass Storage Module

ISDE Property	Value	Description
Name	g_ux_host_class_storage0	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class Mass Storage Lower-Level Modules

Only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the USBX Host Class Mass Storage Source

ISDE Property	Value	Description
Use FileX Stub	Enabled, Disabled Default: Disabled	UX_NO_FILEX Enable this option only in cases where FileX is not available to use.
Maximum number of SCSI logical units	Value must be greater than 0 or empty Default: 1	UX_MAX_HOST_LUN The value represents the maximum number of SCSI logical units represented in the host storage class driver.

Maximum number of storage media instance	Value must be greater than 0 or empty Default: 1	UX_HOST_CLASS_STORAGE_MAX_MEDIA The value represents the maximum number of storage media instances represented in the host storage class driver.
Storage memory size in bytes for FileX used for data transfer	Value must be greater than 512 or empty Default: 1024	UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE The value represents the block of memory in bytes for FileX to use when doing transfers. The value can be changed to save on memory space but should not be smaller than the media sector size. Because USB devices are SCSI devices and there is a great deal of overhead when doing read/writes, it is better to increase it for higher data throughput.

Maximum transfer size in bytes in one BOT data-transport phase	Value must be greater than 512 or empty Default: 1024	UX_HOST_CLASS_STORAGE_MAX_TRANSFER_SIZE The value represents maximum size of memory chunk in bytes to send in one data-transport phase in the Bulk-Only Transport protocol. Large size of data transfer request is split into smaller chunks specified by this configuration. It is better to increase it for higher data throughput. Note: Sufficient numbers of TDs are required, when user changes this parameter for large size(FS more than 8KB and HS more the 64KB) of data transfer. Sufficient number of 'Maximum TDs' needs to be specified in USBX Source module(USBX Source property >> Common >> Maximum TDs). - Maximum TDs for FS mode (maximum data transfer size in bytes /64 bytes = TDs + additional TDs for SCSI wrapper commands). - Maximum TDs for HS mode (maximum data transfer size in bytes /512 bytes = TDs + additional TDs for SCSI wrapper commands).
Stack size for the Mass Storage Class internal thread	Value must be greater than 512 or empty Default: 1024	UX_HOST_CLASS_STORAGE_THREAD_STACK_SIZE The value represents the stack size in bytes for Mass Storage Class internal thread named ux_storage_thread.
Timeout (in milliseconds) for a BOT transfer request	Value must be greater than 0 or empty Default: 100000	UX_HOST_CLASS_STORAGE_TRANSFER_TIMEOUT_IN_MS The value represents timeout value in millisecond for a data transfer request in Bulk-Only Transport protocol.

Timeout (in milliseconds) for the status from a command in the Control/Bulk/Interrupt transport	Value must be greater than 0 or empty Default: 30000	UX_HOST_CLASS_STORAGE_CBI_STATUS_TIMEOUT_IN_MS The value represents timeout value in millisecond for the status from a command, which is returned by the interrupt endpoint in the Control/Bulk/Interrupt transport. The transport is mainly used by storage devices that have very slow read/write commands.
Host storage class dependency on Filex	Enabled, Disabled Default: Disabled	UX_HOST_CLASS_STORAGE_NO_FILEX If enabled, Filex dependency on host storage class is removed.
Show linkage warning	Enabled, Disabled Default: Enabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

Note

The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Host Configuration Instance

ISDE Property	Value	Description
Name	g_ux_host0	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.
Name of generated initialization function	ux_host_hid_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description

Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for full-speed USB.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 part MCUs)	Enable, Disable Default: Disable	Select the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.
Number of Isochronous Pipes Reserved	0, 1, 2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active Low, Active High Default: Active High	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_hcd_hs_0	Module name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux Instance

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	Name must be a valid C symbol.
USBX Pool Memory Size	18432	See section "Azure RTOS USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.
Name of generated initialization function	ux_common_init0	Name of generated initialization function selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

USBX Host Class Mass Storage Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class Mass Storage Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin

PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.48.6 Using the USBX Host Class Mass Storage Module in an Application

The configurator generates processing to create and register the USBX Host Class Mass Storage Module; however, communication must be done after the Host is connected to the host.

The typical steps in using the USBX Host Class Mass Storage Module in an application are:

1. Get the first instance of the connected device with the ux_host_stack_class_instance_get API.

2. Wait until successful.

3. Wait for the device status to become live.

4. Wait for media to be mounted.

5. Access the file on the media with the FileX API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

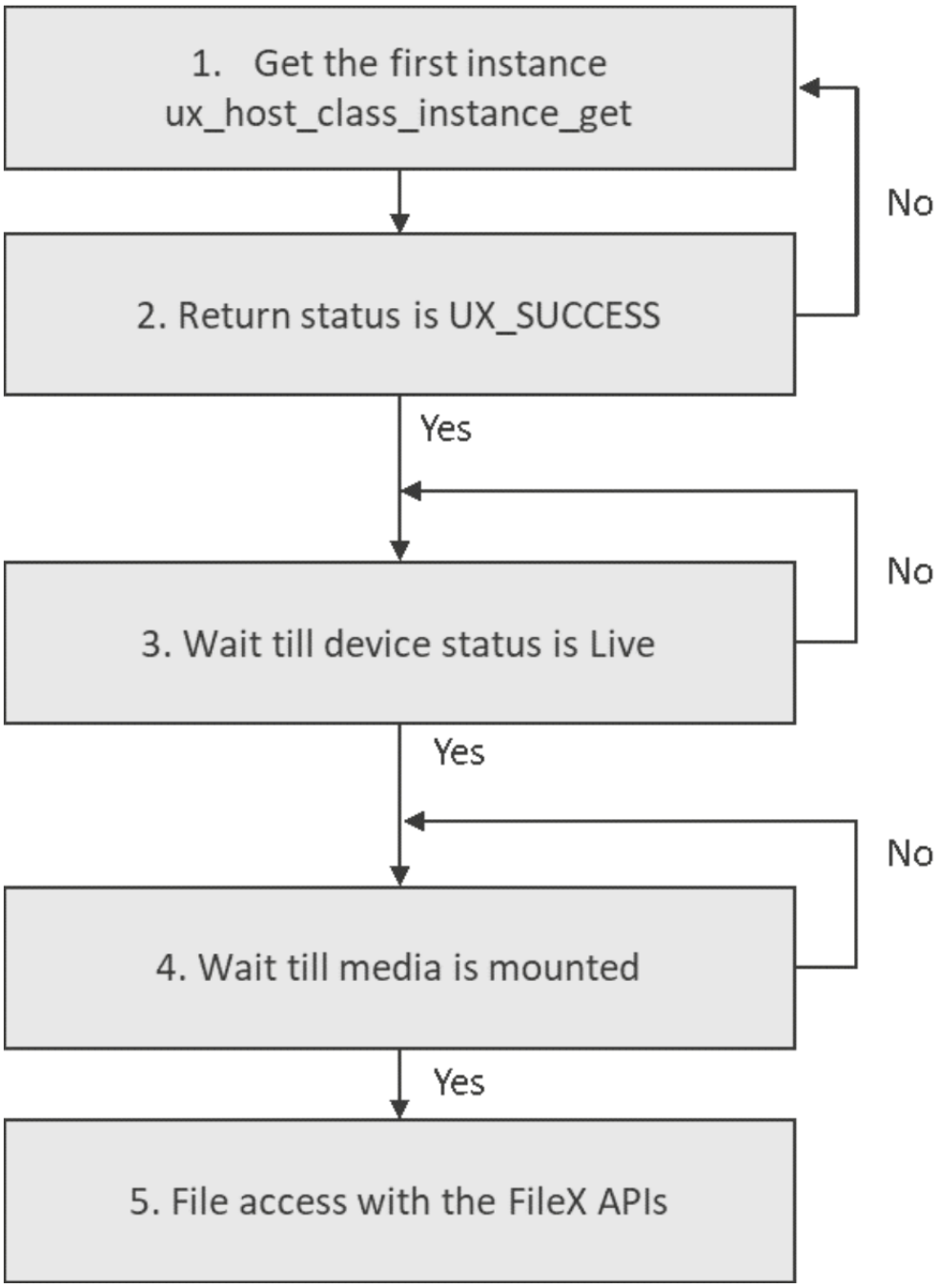


Figure 552: Flow Diagram of a Typical USBX Host Class Mass Storage Module Application

4.3.49 USBX Host Class Video

4.3.49.1 USBX Host Class Video Module Introduction

The USBX™ Host Class Video module provides a high-level API for USBX Host Class Video applications and configures the USBX Host Class Video Source, USBX Host Configuration, USBX

Source and USBX Port Host Controller Device. The USBX Host Class Video module uses the USB peripheral on the Synergy MCU.

Unsupported Features

Multiple instances of the USB device class are not supported on two different physical interfaces.

USBX Host Class Video Module Features

- Supports either a stand-alone hub or functions as part of a compound device such as a keyboard or a monitor.
- Supports either self-powered or bus-powered modes.
- Bus-powered hubs have a maximum of four downstream ports.
- Hubs can be cascaded.
- Up to five hubs can be connected to one another.
- Supports the connection of devices that are either self-powered or bus-powered using less than 100 mA of power.

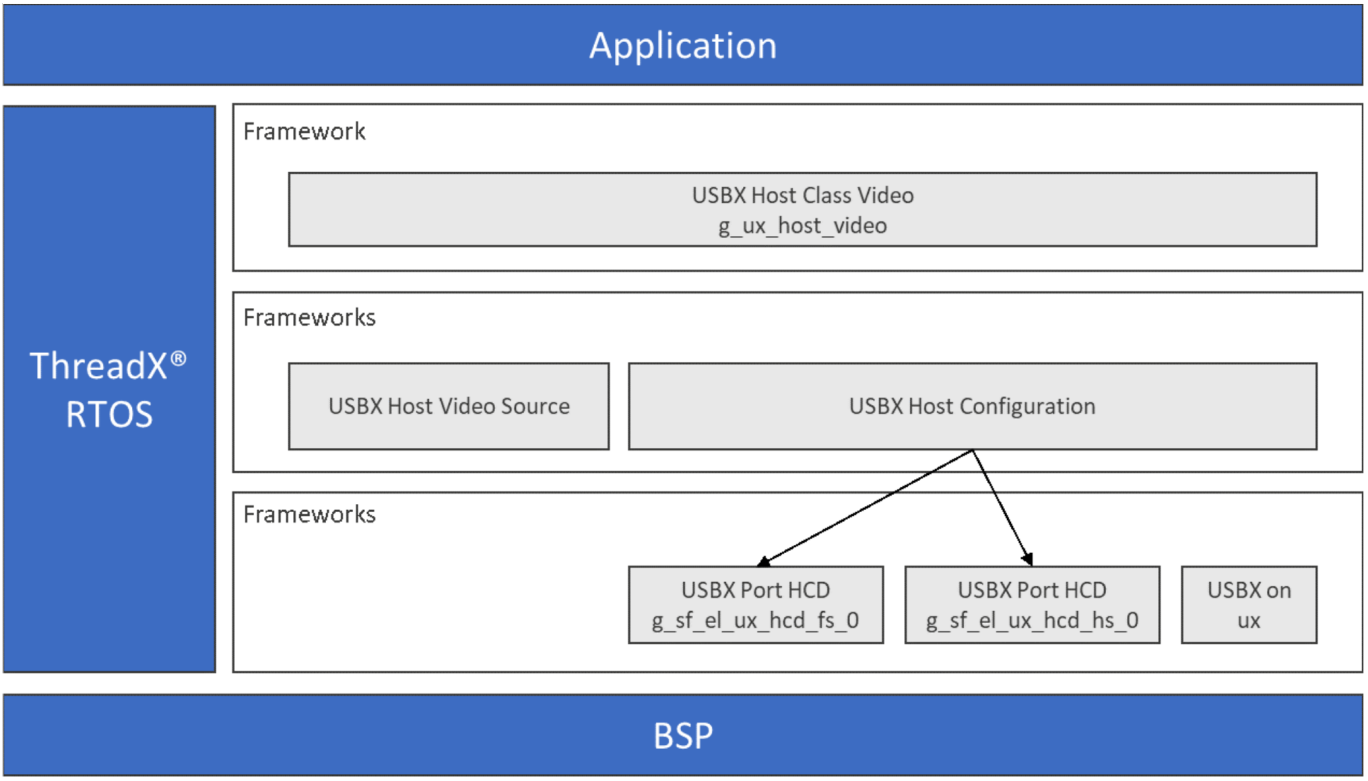


Figure 553: USBX Host Class Video Module Block Diagram

4.3.49.2 USBX Host Class Video Module APIs Overview

The USBX Host Class Video Module does not have an API for the user application.

4.3.49.3 USBX Host Class Video Module Operational Overview

Initialization of USBX Resources

The USBX has its own memory manager. The memory needs to be allocated to the USBX before the host or device side of the USBX is initialized. The USBX memory manager can accommodate systems

where memory can be cached.

Definition of USB Host Controllers

It is required to define at least one USB host controller for USBX to operate in host-mode. The application-initialization file should contain this definition. SSP defines USB host controller when USB host controller driver is added to thread stacks.

Definition of Device Classes

It is required to define one or more device classes(s) with the USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device; one or more classes may be required to drive a USB device depending on the number of interfaces contained in the USB device descriptors.

USB Class Binding

When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device-interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or Human Interface Design (HID).

The class manager contains a list of registered classes from the initialization of the USBX. The class manager will call each class one-at-a-time until one class accepts to manage the interface for that device; each class can only manage one interface. In the case of the USB audio speaker, the class manager will call all the classes for each of the interfaces.

Once a class accepts an interface, a new instance of that class is created; the class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it.

For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

USBX Host Class Video Module Important Operational Notes and Limitations

USBX Host Class Video Module Operational Notes

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code as shown in the following table. You need to set the appropriate memory size in bytes to the USBX Pool Memory Size property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set the total memory size to the property.

Memory (RAM) Requirements for the USBX Memory Pool

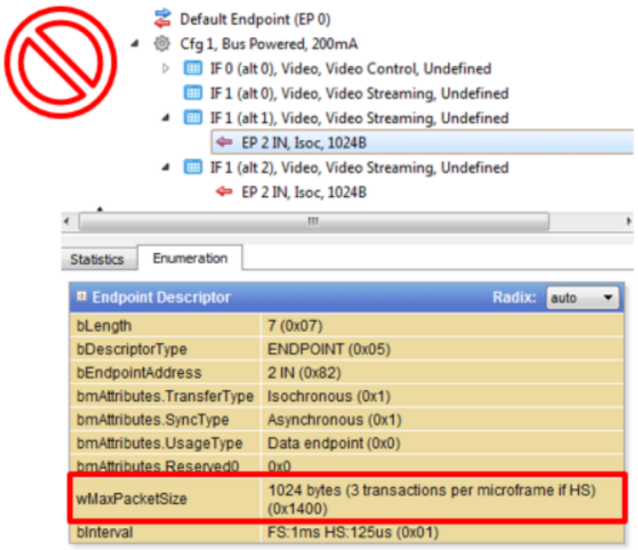
USBX Class	S1 Parts	Other Parts
USBX Host Video (ux_host_class_video)	N/A	35KB

Note

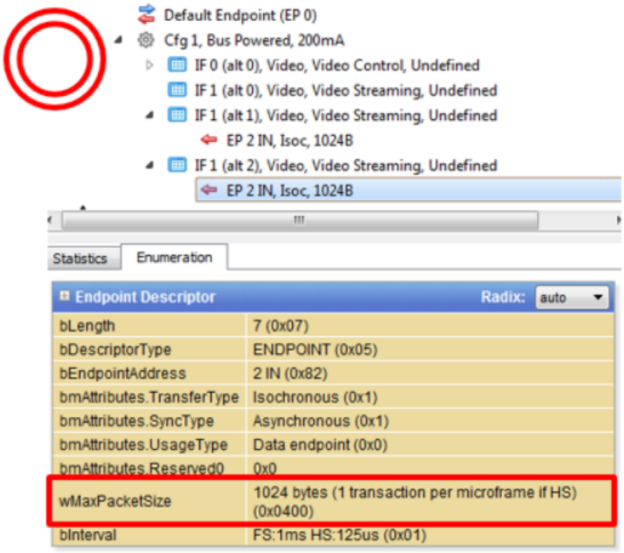
The information shown in the table above is valid if compiled with default USBX configurations.
The memory size of the USBX Classes in the table above are of the pre-built libraries and the following configuration was applied for the builds:UX_THREAD_STACK_SIZE: 512 (bytes) for S1 parts; 2048 (bytes) for the other parts
The USBX Host Video memory pools size may differ according to the type of video device attached to the board. There is no precise formula for memory size. A general idea on a typical USBX Video application is as follows:
(1) Device descriptor: The memory requirement for the device descriptor depends on the video camera in use. Some USB cameras can take 2K memory size while simpler cameras require far less memory for its device descriptors.
(2) End points: around 100 bytes for each end point. The number of end points depends on the device in use. (Could be 5 or more.)
(3) Transfer request buffer: 100 each. The USB control takes one transfer request; the number of transfer requests depends on the application.By default, the USBX video class creates 4 transfer requests.

USBX Host Class Video Module Limitations

To specify an Alt. setting in an interface in the UVC (USB video class) device, you may need to use the USBX API ux_host_class_video_ioctl() instead of ux_host_class_video_start(), specifying UX_HOST_CLASS_VIDEO_IOCTL_CHANNEL_START for the argument "ioctl_function". You also need to specify the argument parameter with setting the member .ux_host_class_video_parameter_channel_bandwidth_selection to match to the wMaxPacketSize of an isochronous endpoint. For instance, if .ux_host_class_video_parameter_channel_bandwidth_selection is set to 1024 for a UVC device with the device descriptor shown in the following figure, the IF1 Alt.1 will not be selected (since it requires high-bandwidth transfer). Instead, IF1 Alt.2 will be selected. For limitations of isochronous transfers, see section "Logic USBX Synergy Port Framework Limitations."



(this will not be selected)



(this will be selected)

Note

Currently, DTC is not supported by the host side driver (only DMAC is supported).

4.3.49.4 Including the USBX Host Class Video Module in an Application

This section describes how to include the USBX Host Class Video Module in an application using the SSP configurator.

Note

It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the USBX Host Class Video Module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

USBX Host Class Video Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ux_host_class_video0 USBX Host Class Video	Threads	New Stack> X-Ware> USBX> Host > Classes > Video > USBX Host Class Video

When the USBX Host Class Video module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

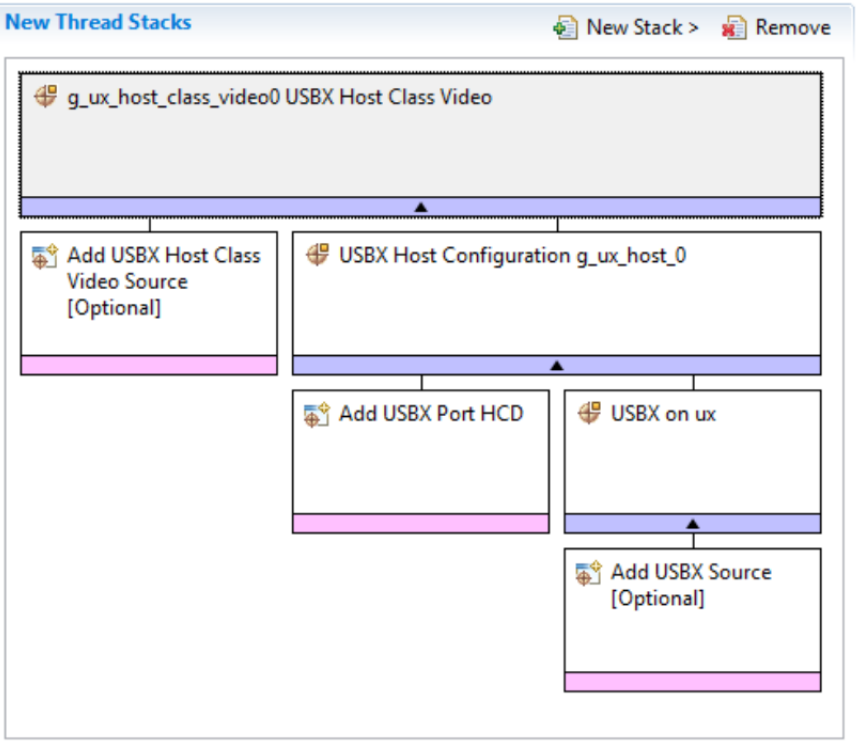


Figure 554: USBX Host Class Video Module Stack

4.3.49.5 Configuring the USBX Host Class Video Module

The USBX Host Class Video Module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

Note

You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the USBX Host Class Video Module

ISDE Property	Value	Description
Class Instance Name	g_ux_host_class_video0	Class instance name.

Note

The example settings and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the USBX Host Class Video Lower-Level Modules

The USBX Host Class Printer Module defines APIs, which are used to interact with the Printer interface. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

USBX Host Class PRINTER Module Summary

Function Name	Example API Call and Description
ux_host_class_printer_read	UINT ux_host_class_printer_read (UX_HOST_CLASS_PRINTER *printer, UCHAR *data_pointer, ULONG requested_length, ULONG *actual_length) This function reads from the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete. A read is allowed only on bi-directional printers.
ux_host_class_printer_write	UINT ux_host_class_printer_write (UX_HOST_CLASS_PRINTER *printer,UCHAR *data_pointer, ULONG requested_length,ULONG *actual_length) This function writes to the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete.
ux_host_class_printer_soft_reset	UINT ux_host_class_printer_soft_reset (UX_HOST_CLASS_PRINTER *printer) This function performs a soft reset to the printer
ux_host_class_printer_status_get	UINT ux_host_class_printer_status_get (UX_HOST_CLASS_PRINTER *printer,ULONG *printer_status) This function obtains the printer status. The printer status is similar to the LPT status (1284 standard).
_ux_host_class_printer_name_get	UINT _ux_host_class_printer_name_get(UX_HOST_CLASS_PRINTER *printer) This function obtains the Device ID string.

Note

for more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Return Values

Name	Description
UX_SUCCESS	This operation was successful.

UX_FUNCTION_NOT_SUPPORTED	Function not supported because the printer is not bi-directional.
UX_TRANSFER_TIMEOUT	Transfer timeout, reading or writing is incomplete, or reset not completed.
UX_MEMORY_INSUFFICIENT	Not enough memory to perform the operation.

Note

Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

USBX Host Class Video Module Clock Configuration

The USB peripheral module uses the UCLK as its clock source; the UCLK should be configured for 48MHz operation. In the SSP configuration window, select the Clocks tab to view the clock-source setting.

USBX Host Class Video Module Pin Configuration

The USB peripheral module uses MCU pins to communicate with external devices. Select I/O pins and configure to the external device requirements. The following table lists the pin selection method within the SSP Configuration Window and the subsequent tables demonstrate the selection process using USB pins as an example.

Note

The selected operation mode determines what peripheral signals available and what MCU pins are required.

USBFS and USBHS Pin Selection Sequence

Resource	ISDE Tab	Pin selection Sequence
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

Note

The selection sequence assumes USBFS0 or USBHS0 is the desired hardware target for the driver.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select device as the Operation Mode
USBDP	USBDP	USBDP pin
USBDM	USBDM	USBDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin

VBUSEN	None	VBUSEN pin
VBUS	None, P407 Default: P407	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID Pin
VCCUSB	VCCUSB	VCCUSB pin
VSSUSB	VSSUSB	VSSUSB pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

USBHS Pin Configuration Settings

Property	Value	Description
Operation Mode	Disabled, Custom, Device, Host, OTG Default: Custom	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP pin
USBHSDM	USBHSDM	USBHSDM pin
OVRCURB	None	OVRCURB pin
OVRCURA	None	OVRCURA pin
VBUSEN	PB00	VBUSEN pin
VBUS	PB01	VBUS pin
EXICEN	None	EXICEN pin
ID	None	ID pin
USBHSRREF	USBHSRREF	USBHSRREF pin
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS pin
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS pin
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS pin
VCCUSBHS	VCCUSBHS	VCCUSBHS pin
VSS1USBHS	VSS1USBHS	VSS1USBHS pin
VSS2USBHS	VSS2USBHS	VSS2USBHS pin

Note

The example settings are for a project using the S7G2 Synergy MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.49.6 Using the USBX Host Class Video Module in an Application

The configurator generates processing to create and register the USBX Host Class Video Module; however, communication must be done after the Host is connected to the host.

The typical steps in using the USBX Host Class Video Module in an application are:

1. Wait until the device is inserted using the tx_event_flags_get API function

2. Set the camera parameters using the ux_host_class_video_frame_parameters_set API function

3. Set the user callback function using the ux_host_class_video_transfer_callback_set API function

4. Get the maximum memory buffer size using the ux_host_class_video_max_payload_get API function

5. Start video transfer using the ux_host_class_video_start API function

6. Add a video buffer to the video device using the ux_host_class_video_transfer_buffer_add API function

7. Wait for a received data frame using the tx_semiphore_get API function

8. Add the video buffer back using the ux_host_class_video_transfer_buffer_add API function

9. Stop the video transfer using the ux_host_class_video_stop API function

These common steps are illustrated in a typical operational flow diagram in the following figure:

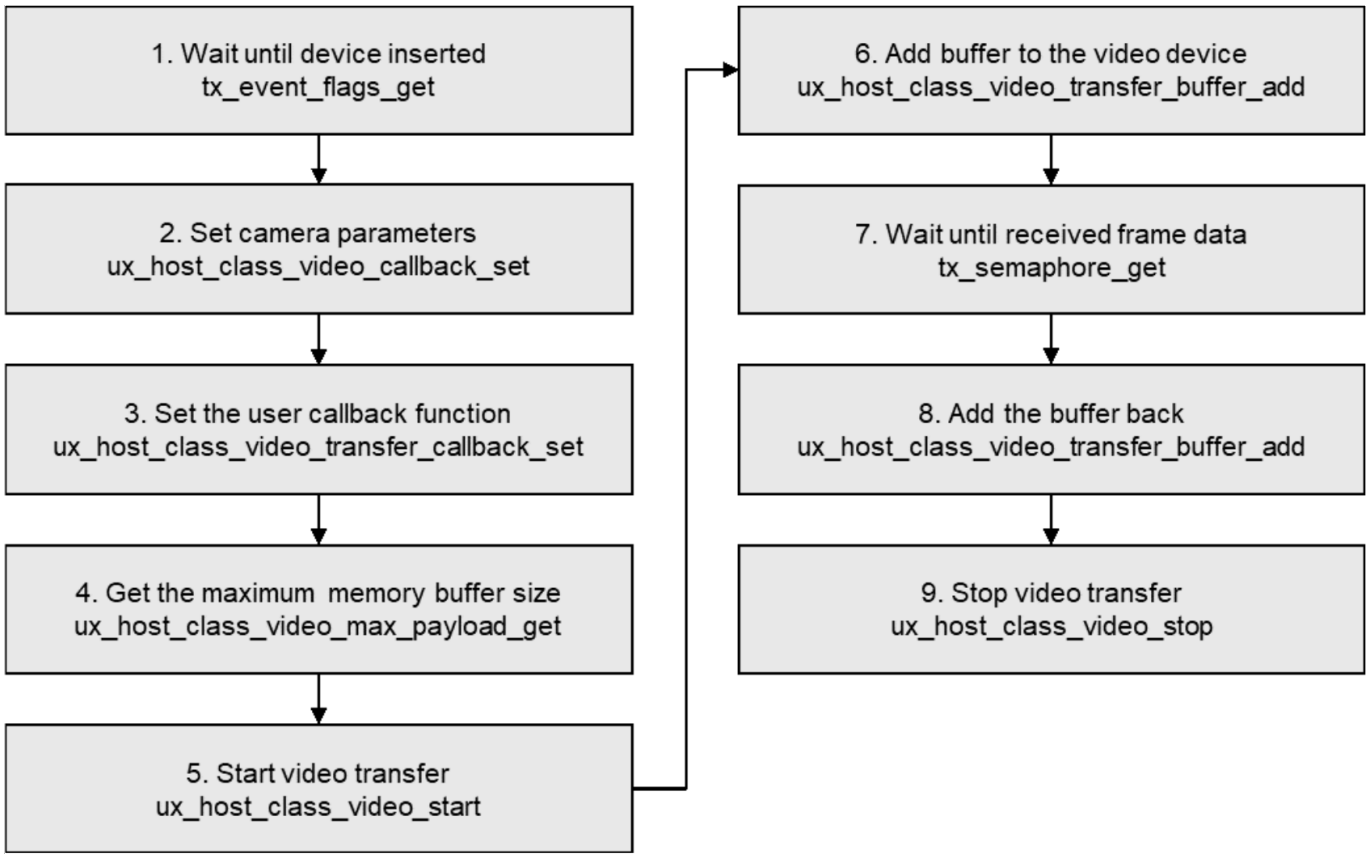


Figure 555: Flow Diagram of a Typical USBX Host Class Video Module Application

Chapter 5 API Reference

This section includes the SSP API Reference for the Framework and HAL level functions.

5.1 Renesas Synergy Software Package Reference

Modules

	Shared
	Framework Interfaces
	Framework Layer
	HAL Interfaces
	HAL Layer

Detailed Description

5.1.1 Shared

Renesas Synergy Software Package Reference

Modules

	Common Error Codes

Detailed Description

Common code shared by SSP drivers

5.1.1.1 Common Error Codes

Renesas Synergy Software Package Reference » Shared

#define	SSP_PARAMETER_NOT_USED(p) (void) ((p))
#define	SSP_CPP_HEADER
enum	ssp_err_t { , **SSP_ERR_ASSERTION** = 1, **SSP_ERR_INVALID_POINTER** = 2, **SSP_ERR_INVALID_ARGUMENT** = 3, **SSP_ERR_INVALID_CHANNEL** = 4, **SSP_ERR_INVALID_MODE** = 5, **SSP_ERR_UNSUPPORTED** = 6, **SSP_ERR_NOT_OPEN** = 7, **SSP_ERR_IN_USE** = 8, **SSP_ERR_OUT_OF_MEMORY** = 9, **SSP_ERR_HW_LOCKED** = 10, **SSP_ERR_IRQ_BSP_DISABLED** = 11, **SSP_ERR_OVERFLOW** = 12, **SSP_ERR_UNDERFLOW** = 13, **SSP_ERR_ALREADY_OPEN** = 14, **SSP_ERR_APPROXIMATION** = 15, **SSP_ERR_CLAMPED** = 16, **SSP_ERR_INVALID_RATE** = 17, **SSP_ERR_ABORTED** = 18, **SSP_ERR_NOT_ENABLED** = 19, **SSP_ERR_TIMEOUT** = 20, **SSP_ERR_INVALID_BLOCKS** = 21, **SSP_ERR_INVALID_ADDRESS** = 22, **SSP_ERR_INVALID_SIZE** = 23, **SSP_ERR_WRITE_FAILED** = 24, **SSP_ERR_ERASE_FAILED** = 25, **SSP_ERR_INVALID_CALL** = 26, **SSP_ERR_INVALID_HW_CONDITION** = 27, **SSP_ERR_INVALID_FACTORY_FLASH** = 28, **SSP_ERR_INVALID_FMI_DATA** = 29, **SSP_ERR_INVALID_STATE** = 30, **SSP_ERR_NOT_ERASED** = 31, **SSP_ERR_SECTOR_RELEASE_FAILED** = 32, **SSP_ERR_INTERNAL** = 100, **SSP_ERR_WAIT_ABORTED** = 101, **SSP_ERR_FRAMING** = 200, **SSP_ERR_BREAK_DETECT** = 201, **SSP_ERR_PARITY** = 202, **SSP_ERR_RXBUF_OVERFLOW** = 203, **SSP_ERR_QUEUE_UNAVAILABLE** = 204, **SSP_ERR_INSUFFICIENT_SPACE** = 205, **SSP_ERR_INSUFFICIENT_DATA** = 206, **SSP_ERR_TRANSFER_ABORTED** = 300, **SSP_ERR_MODE_FAULT** = 301, **SSP_ERR_READ_OVERFLOW** = 302, **SSP_ERR_SPI_PARITY** = 303, **SSP_ERR_OVERRUN** = 304, **SSP_ERR_CLOCK_INACTIVE** = 400, **SSP_ERR_CLOCK_ACTIVE** = 401, **SSP_ERR_STABILIZED** = 402, **SSP_ERR_NOT_STABILIZED** = 403, **SSP_ERR_MAIN_OSC_INACTIVE** = 404, **SSP_ERR_OSC_STOP_DET_ENABLED** = 405, **SSP_ERR_OSC_STOP_DETECTED** = 406, **SSP_ERR_OSC_STOP_CLOCK_ACTIVE** = 407, **SSP_ERR_CLKOUT_EXCEEDED** = 408, **SSP_ERR_USB_MODULE_ENABLED** = 409, **SSP_ERR_HARDWARE_TIMEOUT** = 410, **SSP_ERR_PE_FAILURE**

= 500, **SSP_ERR_CMD_LOCKED** = 501, **SSP_ERR_FCLK** = 502,
SSP_ERR_INVALID_LINKED_ADDRESS = 503,
SSP_ERR_INVALID_CAC_REF_CLOCK = 600,
SSP_ERR_CLOCK_GENERATION = 1000,
SSP_ERR_INVALID_TIMING_SETTING = 1001,
SSP_ERR_INVALID_LAYER_SETTING = 1002,
SSP_ERR_INVALID_ALIGNMENT = 1003,
SSP_ERR_INVALID_GAMMA_SETTING = 1004,
SSP_ERR_INVALID_LAYER_FORMAT = 1005,
SSP_ERR_INVALID_UPDATE_TIMING = 1006,
SSP_ERR_INVALID_CLUT_ACCESS = 1007,
SSP_ERR_INVALID_FADE_SETTING = 1008, **SSP_ERR_JPEG_ERR** = 1100,
SSP_ERR_JPEG_SOI_NOT_DETECTED = 1101,
SSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED = 1102,
SSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT = 1103,
SSP_ERR_JPEG_SOF_ACCURACY_ERROR = 1104,
SSP_ERR_JPEG_DQT_ACCURACY_ERROR = 1105,
SSP_ERR_JPEG_COMPONENT_ERROR1 = 1106,
SSP_ERR_JPEG_COMPONENT_ERROR2 = 1107,
SSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED = 1108,
SSP_ERR_JPEG_SOS_NOT_DETECTED = 1109,
SSP_ERR_JPEG_EOI_NOT_DETECTED = 1110,
SSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR = 1111, **SSP_ERR_JPEG_IMAGE_SIZE_ERROR** = 1112,
SSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR = 1113,
SSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR = 1114,
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH = 1115,
SSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE = 1116,
SSP_ERR_CALIBRATE_FAILED = 1200,
SSP_ERR_IP_HARDWARE_NOT_PRESENT = 1400,
SSP_ERR_IP_UNIT_NOT_PRESENT = 1401,
SSP_ERR_IP_CHANNEL_NOT_PRESENT = 1402,
SSP_ERR_NO_MORE_BUFFER = 2000,
SSP_ERR_ILLEGAL_BUFFER_ADDRESS = 2001,
SSP_ERR_INVALID_WORKBUFFER_SIZE = 2002,
SSP_ERR_INVALID_MSG_BUFFER_SIZE = 2003,
SSP_ERR_TOO_MANY_BUFFERS = 2004,
SSP_ERR_NO_SUBSCRIBER_FOUND = 2005,
SSP_ERR_MESSAGE_QUEUE_EMPTY = 2006,
SSP_ERR_MESSAGE_QUEUE_FULL = 2007,
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS = 2008,
SSP_ERR_BUFFER_RELEASED = 2009,
SSP_ERR_D2D_ERROR_INIT = 3000,
SSP_ERR_D2D_ERROR_DEINIT = 3001,
SSP_ERR_D2D_ERROR_RENDERING = 3002,
SSP_ERR_D2D_ERROR_SIZE = 3003, **SSP_ERR_QUEUE_FULL** = 10000, **SSP_ERR_QUEUE_EMPTY** = 10001,
SSP_ERR_CTSU_SC_OVERFLOW = 0x8010,
SSP_ERR_CTSU_RC_OVERFLOW = 0x8020,
SSP_ERR_CTSU_ICOMP = 0x8040,
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED = 0x8080,
SSP_ERR_CTSU_SAFETY_CHECK_FAILED = 0x8100,
SSP_ERR_CTSU_SCANNING = 0x8200,
SSP_ERR_CTSU_NOT_GET_DATA = 0x8201,

SSP_ERR_CTSU_INCOMPLETE_TUNING = 0x8202,
SSP_ERR_CTSU_DIAG_NOT_YET = 6003,
SSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,
SSP_ERR_CTSU_DIAG_CCO_HIGH = 6005,
SSP_ERR_CTSU_DIAG_CCO_LOW = 6006,
SSP_ERR_CTSU_DIAG_SSCG = 6007, **SSP_ERR_CTSU_DIAG_DAC** = 6008, **SSP_ERR_CARD_INIT_FAILED** = 40000,
SSP_ERR_CARD_NOT_INSERTED = 40001,
SSP_ERR_SDHI_FAILED = 40002, **SSP_ERR_READ_FAILED** = 40003, **SSP_ERR_CARD_NOT_READY** = 40004,
SSP_ERR_CARD_WRITE_PROTECTED = 40005,
SSP_ERR_TRANSFER_BUSY = 40006,
SSP_ERR_MEDIA_FORMAT_FAILED = 50000,
SSP_ERR_MEDIA_OPEN_FAILED = 50001,
SSP_ERR_CAN_DATA_UNAVAILABLE = 60000,
SSP_ERR_CAN_MODE_SWITCH_FAILED = 60001,
SSP_ERR_CAN_INIT_FAILED = 60002,
SSP_ERR_CAN_TRANSMIT_NOT_READY = 60003,
SSP_ERR_CAN_RECEIVE_MAILBOX = 60004,
SSP_ERR_CAN_TRANSMIT_MAILBOX = 60005,
SSP_ERR_CAN_MESSAGE_LOST = 60006,
SSP_ERR_WIFI_CONFIG_FAILED = 70000,
SSP_ERR_WIFI_INIT_FAILED = 70001,
SSP_ERR_WIFI_TRANSMIT_FAILED = 70002,
SSP_ERR_WIFI_INVALID_MODE = 70003, **SSP_ERR_WIFI_FAILED** = 70004, **SSP_ERR_WIFI_WPS_MULTIPLE_PB_SESSIONS** = 70005,
SSP_ERR_WIFI_WPS_M2D_RECEIVED = 70006,
SSP_ERR_WIFI_WPS_AUTHENTICATION_FAILED = 70007,
SSP_ERR_WIFI_WPS_CANCELLED = 70008,
SSP_ERR_WIFI_WPS_INVALID_PIN = 70009,
SSP_ERR_CELLULAR_CONFIG_FAILED = 80000,
SSP_ERR_CELLULAR_INIT_FAILED = 80001,
SSP_ERR_CELLULAR_TRANSMIT_FAILED = 80002,
SSP_ERR_CELLULAR_FW_UPTODATE = 80003,
SSP_ERR_CELLULAR_FW_UPGRADE_FAILED = 80004,
SSP_ERR_CELLULAR_FAILED = 80005,
SSP_ERR_CELLULAR_INVALID_STATE = 80006,
SSP_ERR_CELLULAR_REGISTRATION_FAILED = 80007,
SSP_ERR_BLE_FAILED = 90001, **SSP_ERR_BLE_INIT_FAILED** = 90002, **SSP_ERR_BLE_CONFIG_FAILED** = 90003,
SSP_ERR_BLE_PRF_ALREADY_ENABLED = 90004,
SSP_ERR_BLE_PRF_NOT_ENABLED = 90005,
SSP_ERR_CRYPTO_CONTINUE = 0x10000,
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT = 0x10001,
SSP_ERR_CRYPTO_SCE_FAIL = 0x10002,
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX = 0x10003,
SSP_ERR_CRYPTO_SCE_RETRY = 0x10004,
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL = 0x10005,
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN = 0x10006,
SSP_ERR_CRYPTO_NOT_OPEN = 0x10007,
SSP_ERR_CRYPTO_UNKNOWN = 0x10008,
SSP_ERR_CRYPTO_NULL_POINTER = 0x10009,
SSP_ERR_CRYPTO_NOT_IMPLEMENTED = 0x1000a,
SSP_ERR_CRYPTO_RNG_INVALID_PARAM = 0x1000b,

	SSP_ERR_CRYPTO_RNG_FATAL_ERROR = 0x1000c, **SSP_ERR_CRYPTO_INVALID_SIZE** = 0x1000d, **SSP_ERR_CRYPTO_INVALID_STATE** = 0x1000e, **SSP_ERR_CRYPTO_ALREADY_OPEN** = 0x1000f, **SSP_ERR_CRYPTO_INSTALL_KEY_FAILED** = 0x10010, **SSP_ERR_CRYPTO_AUTHENTICATION_FAILED** = 0x10011, **SSP_ERR_CRYPTO_COMMON_NOT_OPENED** = 0x20000, **SSP_ERR_CRYPTO_HAL_ERROR** = 0x20001, **SSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH** = 0x20002, **SSP_ERR_CRYPTO_BUF_OVERFLOW** = 0x20003, **SSP_ERR_CRYPTO_INVALID_OPERATION_MODE** = 0x20004, **SSP_ERR_MESSAGE_TOO_LONG** = 0x20005, **SSP_ERR_RSA_DECRYPTION_ERROR** = 0x20006 }

enum	ssp_command_t {
	SSP_COMMAND_GET_SECTOR_COUNT =1,
	SSP_COMMAND_GET_SECTOR_SIZE =2,
	SSP_COMMAND_GET_BLOCK_SIZE =3,
	SSP_COMMAND_CTRL_ERASE_SECTOR =4,
	SSP_COMMAND_GET_WRITE_PROTECTED =5,
	SSP_COMMAND_SET_BLOCK_SIZE =6,
	SSP_COMMAND_GET_SECTOR_RELEASE =7,
	SSP_COMMAND_CTRL_SECTOR_RELEASE =8
	}

Detailed Description

All SSP code at every layer shares these common error codes.

Macro Definition Documentation

◆ SSP_CPP_HEADER

#define SSP_CPP_HEADER
Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.

◆ SSP_PARAMETER_NOT_USED

#define SSP_PARAMETER_NOT_USED (p) (void) ((p))
This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.

Enumeration Type Documentation

◆ ssp_command_t

enum ssp_command_t
ioctl commands.

◆ ssp_err_t

enum ssp_err_t
Common error codes

5.1.2 Framework Interfaces

Renesas Synergy Software Package Reference

Modules

	ADC Periodic Framework Interface RTOS-integrated ADC Periodic Framework Interface.
	Audio Framework Interface RTOS-integrated Audio Framework Interface.
	Audio Playback Framework Interface RTOS-integrated Audio Playback Framework Interface.
	Audio Recording Framework Interface RTOS-integrated Audio Recording Framework Interface.
	SF BLE Framework Interface RTOS-integrated SF BLE Framework Interface.
	SF BLE On-Board Profile Framework Interface RTOS-integrated SF BLE On-Board Profile Framework Interface.
	SF BLE Alert Notification Profile Framework Interface RTOS-integrated SF BLE Alert Notification Profile Framework Interface.

	SF BLE Battery Service Profile Framework Interface RTOS-integrated SF BLE Battery Service Profile Framework Interface.
	SF BLE Blood Pressure Profile Framework Interface RTOS-integrated SF BLE Blood Pressure Profile Framework Interface.
	SF BLE Current Time Service Profile Framework Interface RTOS-integrated SF BLE Current Time Service Profile Framework Interface.
	SF BLE Find Me Profile Framework Interface RTOS-integrated SF BLE Find Me Profile Framework Interface.
	SF BLE HID Over GATT Profile Framework Interface RTOS-integrated SF BLE HID Over GATT Profile Framework Interface.
	SF BLE Heart Rate Profile Framework Interface RTOS-integrated SF BLE Heart Rate Profile Framework Interface.
	SF BLE Health Thermometer Profile Framework Interface RTOS-integrated SF BLE Health Thermometer Profile Framework Interface.
	SF BLE Immediate Alert Profile Framework Interface RTOS-integrated SF BLE Immediate Alert Profile Framework Interface.
	SF BLE Next DST Change Service Profile Framework Interface RTOS-integrated SF BLE Next DST Change Service Profile Framework Interface.
	SF BLE Phone Alert Status Profile Framework Interface RTOS-integrated SF BLE Phone Alert Status Profile Framework Interface.

	SF BLE Proximity Profile Framework Interface RTOS-integrated SF BLE Proximity Profile Framework Interface.
	SF BLE Reference Time Update Service Profile Framework Interface RTOS-integrated SF BLE Reference Time Update Service Profile Framework Interface.
	SF BLE Scan Parameters Service Profile Framework Interface RTOS-integrated SF BLE Scan Parameters Service Profile Framework Interface.
	SF BLE Time Information Profile Framework Interface RTOS-integrated SF BLE Time Information Profile Framework Interface.
	Block Media Framework Interface RTOS-integrated File system Interface to access Synergy block media devices.
	SF CELLULAR Framework Interface RTOS-integrated SF CELUULAR Framework Interface.
	SF CELLULAR NSAL Framework Interface RTOS-integrated SF CELLULAR NSAL Framework Interface.
	SF Socket CELLULAR Framework Interface RTOS-integrated SF Socket Cellular Framework Interface.
	Communications Framework Interface RTOS-integrated communications Framework Interface.
	Console Framework Interface RTOS-integrated Console Framework Interface.
	SSP Crypto Framework Common Module Interface

	Interface definition for Synergy Crypto Framework module.
	SSP Crypto Cipher Framework Interface Interface definition for Synergy Crypto Cipher Framework module.
	SSP Crypto HASH Framework Interface Interface definition for Synergy Crypto HASH Framework module.
	SSP Crypto Key Framework Interface Interface definition for Synergy Crypto Key Framework module.
	SSP Crypto Key Installation Framework Interface Interface definition for Synergy Crypto Key Installation Framework module.
	SSP Crypto Signature Framework Interface Interface definition for Synergy Crypto Signature Framework module.
	SSP Crypto TRNG Framework Interface Interface definition for Synergy Crypto TRNG Framework module.
	GUIX Interface Interface definition for Adapting Microsoft GUIX for Synergy graphics drivers.
	External IRQ Framework Interface RTOS-integrated External IRQ Framework Interface.
	I2C Framework RTOS-integrated I2C Framework Interface.
	JPEG Decode Framework Interface RTOS-integrated JPEG Decode Framework Interface.
	Memory interface

	Interface for Memory API.
	Messaging Framework Interface RTOS-integrated Messaging Framework Interface.
	Power Profiles V2 Framework Interface Power Profiles Framework Interface.
	SF Socket WIFI Framework Interface RTOS-integrated SF Socket WIFI Framework Interface.
	SPI Framework Interface RTOS-integrated SPI Framework Interface.
	Thread Monitor Framework Interface RTOS-integrated Framework Interface for monitoring of threads.
	CTSU v2 Framework Interface CTSU v2 Framework Interface.
	Touch chip Interface RTOS-integrated Touch chip Interface.
	Touch Panel Framework Interface RTOS-integrated Touch Panel Framework Interface.
	SF WIFI Framework Interface RTOS-integrated SF WIFI Framework Interface.
	SF WIFI NSAL Interface RTOS-integrated SF WIFI NSAL Framework Interface.
	SF WIFI On-Chip Stack Interface RTOS-integrated SF WIFI On-Chip Stack Interface.

	SF WIFI QCA4010 Framework Interface RTOS-integrated SF WIFI QCA4010 Framework Interface.
	SF WIFI QCA4010 On-Chip Interface RTOS-integrated SF Socket Wifi Framework Interface.
	SF Socket WIFI Framework Interface RTOS-integrated SF Socket WIFI Framework Interface.
	SF WIFI NSAL on NetX NetX NSAL interface implementation header file.
	BLE Framework Interface on RL78G1D RTOS-integrated BLE Interface Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces:
	Cellular Framework Example using Quectel CATM1 API SF_CELLULAR Framework API on Quectel CATM1.
	BSD Socket over Quectel CATM1 on-chip stack API SF_CELLULAR Socket Framework API on Quectel CATM1.
	Cellular Framework Example using RYZ014CATM1 API SF_CELLULAR Framework API on RYZ014CATM1.
	SF CELLULAR Common Interface SF_Cellular Framework API Common Code.
	BSD Socket over RYZ014CATM1 on-chip stack API SF_CELLULAR Socket Framework API on RYZ014CATM1.

Detailed Description

The Framework Interface provides common APIs for functional Framework Layer applications. The

Framework Interfaces can be implemented by one or more Framework Layer drivers.

5.1.2.1 ADC Periodic Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated ADC Periodic Framework Interface. More...

Data Structures

struct	sf_adc_periodic_callback_args_t
struct	sf_adc_periodic_cfg_t
struct	sf_adc_periodic_api_t
struct	sf_adc_periodic_instance_t

Macros

#define	SF_ADC_PERIODIC_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_adc_periodic_ctrl_t

Enumerations

enum	sf_adc_periodic_event_t { SF_ADC_PERIODIC_EVENT_NEW_DATA }

Detailed Description

RTOS-integrated ADC Periodic Framework Interface.

Summary

This is a ThreadX aware generic Periodic ADC sampling framework intended to be used to sample the ADC at periodic intervals, buffer the specified number of samples and then notify the application. The driver will use hardware triggers to allow for time-synchronous sampling. After initial configuration and the scan process is started, the framework uses a hardware timer to trigger an ADC scan in one-shot mode. Each scan can consist of one or more channels. When each scan is completed the ADC interrupt is intercepted by the DTC which moves the result of the scan into the user buffer. Each scan is defined as a sampling iteration and the number of samples generated for each scan will be equal to the number of channels if the channels are sequential eg: channels 1, 2, 3, 4. If the channels are not in sequence, eg: channels 1, 3, 4, 5, then the samples generated by each scan will also include data from the unused channels in between. Thus the second example here will result in 5 samples being stored to the user buffer each time even though only 4 channels have been configured for usage.

The user specifies the total number of sample iterations that need to occur before being notified. When the specified number of sampling iterations have occurred and the data for each iteration has been stored into the user buffer, the user is notified via the callback function with an index for the valid data in the buffer and an event indicating that sampling for the specified number of iterations is complete. Unless the user stops the scan process using the stop API call, the scan will continue to be triggered by the timer and data will be written into the user buffer which is treated by the framework as a circular buffer. For this reason, the buffer length must be at least twice the total number of samples that will be generate after all the iterations are completed. In the second example, where there are 5 samples generated for each iteration, if the sample count is set to 3, this will result in 15 samples being available in the buffer before the callback is called. Thus in this example, the buffer length must be set to 30 or larger. The name and length of the buffer is specified via the framework configuration structure.

Implemented by: ADC Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

ADC Periodic Framework Interface description: ADC Periodic Framework

Macro Definition Documentation

◆ SF_ADC_PERIODIC_API_VERSION_MAJOR

#define SF_ADC_PERIODIC_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_adc_periodic_ctrl_t

typedef void sf_adc_periodic_ctrl_t
ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. **Implemented as** ◦ sf_adc_periodic_instance_ctrl_t

Enumeration Type Documentation

◆ sf_adc_periodic_event_t

enum sf_adc_periodic_event_t	
Options for the callback events.	
Enumerator	
SF_ADC_PERIODIC_EVENT_NEW_DATA	New data is available in the buffer.

sf_adc_periodic_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » ADC Periodic Framework Interface

#include <sf_adc_periodic_api.h>

Data Fields

sf_adc_periodic_event_t	event Periodic ADC callback event.
uint32_t	buffer_index Index to the buffer where the new data is stored.
void const *	p_context Placeholder for user data.
adc_data_size_t *	p_data_buffer Pointer to the buffer that will store the samples.
uint32_t	num_new_samples Number of new samples in the data buffer.

Detailed Description

ADC callback arguments definitions

The documentation for this struct was generated from the following file:

- sf_adc_periodic_api.h

sf_adc_periodic_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » ADC Periodic Framework Interface

#include <sf_adc_periodic_api.h>

Data Fields

adc_instance_t const *const	p_lower_lvl_adc Pointer to the ADC instance.
timer_instance_t const *const	p_lower_lvl_timer Pointer to the Timer instance.
transfer_instance_t const *const	p_lower_lvl_transfer Pointer to the Transfer instance.
adc_data_size_t *	p_data_buffer Pointer to the buffer that will store the samples.
bool	lower_level Used to check lower level driver (0 for ADC and 1 for SDADC)
uint32_t	data_buffer_length Length of the data buffer that will store the samples.
uint32_t	sample_count Samples per channel to be buffered before notifying the app.
elc_event_t	scan_trigger

	The hardware trigger that starts the ADC scan.
void(*	p_callback)(sf_adc_periodic_callback_args_t *p_args) Callback function.
void const *	p_context Placeholder for user data.
void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

Configuration for RTOS integrated ADC driver

The documentation for this struct was generated from the following file:

- sf_adc_periodic_api.h

sf_adc_periodic_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » ADC Periodic Framework Interface

#include <sf_adc_periodic_api.h>

Data Fields

ssp_err_t(*	open)(sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)
ssp_err_t(*	start)(sf_adc_periodic_ctrl_t *const p_ctrl)
ssp_err_t(*	stop)(sf_adc_periodic_ctrl_t *const p_ctrl)
ssp_err_t(*	close)(sf_adc_periodic_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Framework Periodic ADC API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_adc_periodic_api_t::close) (sf_adc_periodic_ctrl_t *const p_ctrl)

Releases channel mutex and closes channel at HAL layer.

Parameters

[in]	p_ctrl	Pointer to control block set in SF_ADC_PERIODIC_Open.

◆ open

ssp_err_t(* sf_adc_periodic_api_t::open) (sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)

Acquires mutex, then initializes driver at the HAL layer

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. Elements initialized here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ start

ssp_err_t(* sf_adc_periodic_api_t::start) (sf_adc_periodic_ctrl_t *const p_ctrl)

Starts the scan.

Warning

The driver will enable the ADC to be triggered via timer event; there will be a time delay from the time this function is called to the time the hardware timer count expires and triggers the scan.

Parameters

[in]	p_ctrl	Pointer to control block set in SF_ADC_PERIODIC_Open.

◆ stop

ssp_err_t(* sf_adc_periodic_api_t::stop) (sf_adc_periodic_ctrl_t *const p_ctrl)

Stops the hardware trigger (timer) from triggering any more ADC scans.

Parameters

[in]	p_ctrl	Pointer to control block set in SF_ADC_PERIODIC_Open.

◆ versionGet

ssp_err_t(* sf_adc_periodic_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- sf_adc_periodic_api.h

sf_adc_periodic_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » ADC Periodic Framework Interface

#include <sf_adc_periodic_api.h>

Data Fields

sf_adc_periodic_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_adc_periodic_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.

sf_adc_periodic_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_adc_periodic_api.h

5.1.2.2 Audio Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Audio Framework Interface. More...

Data Structures

struct	sf_audio_playback_data_t
struct	sf_audio_playback_common_cfg_t
struct	sf_audio_playback_cfg_t
struct	sf_audio_playback_api_t
struct	sf_audio_playback_instance_t

Macros

#define	SF_AUDIO_PLAYBACK_API_VERSION_MAJOR (2U)
#define	SF_AUDIO_PLAYBACK_MESSAGE_WORDS ((sizeof(sf_message_payload_audio_t) + 3) / 4)
#define	SF_AUDIO_PLAYBACK_MAX_VOLUME (255)

Typedefs

typedef void	sf_audio_playback_common_ctrl_t
typedef void	sf_audio_playback_ctrl_t

Enumerations

enum sf_audio_playback_status_t {
SF_AUDIO_PLAYBACK_STATUS_STOPPED,
SF_AUDIO_PLAYBACK_STATUS_PAUSED,
SF_AUDIO_PLAYBACK_STATUS_PLAYING,
SF_AUDIO_PLAYBACK_STATUS_WAITING }

Detailed Description

RTOS-integrated Audio Framework Interface.

Summary

The Audio Interface is a ThreadX-aware Audio Framework Interface. The Interface is implemented by the Audio Framework using the Timer driver, the Transfer driver, and a choice of the following drivers for playback: DAC, PWM (to be implemented), or I2S (to be implemented).

Interfaces used:

- Transfer Interface
- DAC Interface
- Timer Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Audio Framework Interface description: Audio Playback Framework

Macro Definition Documentation

◆ SF_AUDIO_PLAYBACK_API_VERSION_MAJOR

#define SF_AUDIO_PLAYBACK_API_VERSION_MAJOR (2U)
Version of the API defined in this file

◆ SF_AUDIO_PLAYBACK_MAX_VOLUME

#define SF_AUDIO_PLAYBACK_MAX_VOLUME (255)
Macro defining the maximum volume.

◆ SF_AUDIO_PLAYBACK_MESSAGE_WORDS

#define SF_AUDIO_PLAYBACK_MESSAGE_WORDS ((sizeof(sf_message_payload_audio_t) + 3) / 4)
Audio playback message size in 4 byte words, rounded up.

Typedef Documentation

◆ sf_audio_playback_common_ctrl_t

typedef void sf_audio_playback_common_ctrl_t

Audio playback common control block. Allocate an instance specific control block to pass to sf_audio_playback_api_t::open in sf_audio_playback_cfg_t.

Implemented as

- sf_audio_playback_common_instance_ctrl_t

◆ sf_audio_playback_ctrl_t

typedef void sf_audio_playback_ctrl_t

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls.

Implemented as

- sf_audio_playback_instance_ctrl_t

Enumeration Type Documentation

◆ sf_audio_playback_status_t

enum sf_audio_playback_status_t

Audio playback status.

Enumerator	
SF_AUDIO_PLAYBACK_STATUS_STOPPED	Stream is available to be used.
SF_AUDIO_PLAYBACK_STATUS_PAUSED	Stream is paused.
SF_AUDIO_PLAYBACK_STATUS_PLAYING	Stream is playing data.
SF_AUDIO_PLAYBACK_STATUS_WAITING	Stream is between packets, waiting for the next data message.

sf_audio_playback_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Framework Interface

#include <sf_audio_playback_api.h>

Data Fields

sf_message_header_t	header Required common members of messaging framework payloads.
sf_audio_playback_data_type_t	type Data type. Must be uncompressed.
uint32_t	size_bytes Size of data in bytes.
void const *	p_data Pointer to data. Data start address must be 4-byte aligned.
UINT	loop_timeout
bool	stream_end

Detailed Description

Audio data for playback.

Field Documentation

◆ loop_timeout

UINT sf_audio_playback_data_t::loop_timeout
ThreadX timeout, select TX_NO_WAIT to play the entire buffer once, TX_WAIT_FOREVER to loop until SF_AUDIO_PLAYBACK_Pause is called from another thread, or any timeout value from (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts to loop until the tick counts expire.

◆ stream_end

bool sf_audio_playback_data_t::stream_end
This releases ownership of the stream and allows other threads to post data. Set to true if not more data will be sent as a part of this logical bitstream. Set to false if more packets are being prepared.

The documentation for this struct was generated from the following file:

- sf_audio_playback_api.h

sf_audio_playback_common_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Framework Interface

#include <sf_audio_playback_api.h>

Data Fields

UINT	priority Priority of the audio playback thread.
sf_audio_playback_hw_instance_t const *	p_lower_lvl_hw Hardware instance.
sf_message_instance_t const *	p_message
TX_QUEUE *	p_queue
void const *	p_extend

Detailed Description

Common configuration for RTOS integrated audio framework. Shared by all streams.

Field Documentation

◆ p_extend

void const* sf_audio_playback_common_cfg_t::p_extend
Implementation specific extension configuration.

◆ p_message

sf_message_instance_t const* sf_audio_playback_common_cfg_t::p_message
Pointer to messaging framework instance used to post audio messages.

◆ p_queue

TX_QUEUE* sf_audio_playback_common_cfg_t::p_queue
Pointer to the messaging framework queue specified for this audio stream. Must be subscribed to the SF_MESSAGE_EVENT_CLASS_AUDIO event class.

The documentation for this struct was generated from the following file:

- sf_audio_playback_api.h

sf_audio_playback_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Framework Interface

#include <sf_audio_playback_api.h>

Data Fields

void(*	p_callback)(sf_message_callback_args_t *p_args)
sf_audio_playback_common_ctrl_t *	p_common_ctrl
sf_audio_playback_common_cfg_t const *	p_common_cfg
uint8_t	class_instance Class instance used to identify the stream to the messaging framework.

Detailed Description

Per stream configuration for RTOS integrated audio framework.

Field Documentation

◆ p_callback

void(* sf_audio_playback_cfg_t::p_callback) (sf_message_callback_args_t *p_args)
Callback called when playback of a buffer passed to sf_audio_playback_api_t::start is complete. Set to NULL for no callback.

◆ p_common_cfg

sf_audio_playback_common_cfg_t const* sf_audio_playback_cfg_t::p_common_cfg
Pointer to common configurations shared by all streams using the same hardware.

◆ p_common_ctrl

sf_audio_playback_common_ctrl_t* sf_audio_playback_cfg_t::p_common_ctrl
Pointer to the hardware control block used by this stream.

The documentation for this struct was generated from the following file:

- sf_audio_playback_api.h

sf_audio_playback_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Framework Interface

#include <sf_audio_playback_api.h>

Data Fields

ssp_err_t(*	open)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg) Configure the audio framework by creating a thread for audio playback and configuring HAL layer drivers used. This function must be called before any other audio functions. More...
ssp_err_t(*	close)(sf_audio_playback_ctrl_t *const p_ctrl) The close API handles cleans up internal driver data. More...

ssp_err_t(*	start)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout) Play audio. Currently only 16-bit mono PCM buffers are supported. More...
ssp_err_t(*	pause)(sf_audio_playback_ctrl_t *const p_ctrl) Pause audio playback. This stops the peripheral that triggers the DMA/DTC transfer and posts a flag to notify SF_AUDIO_PLAYBACK_Start() to pause any playback in progress. More...
ssp_err_t(*	stop)(sf_audio_playback_ctrl_t *const p_ctrl) Stop audio playback. Causes SF_AUDIO_PLAYBACK_Start() halt playback and return. More...
ssp_err_t(*	resume)(sf_audio_playback_ctrl_t *const p_ctrl) Resume audio playback. Posts a flag to notify SF_AUDIO_PLAYBACK_Start() to restart the peripheral that triggers the DMA/DTC transfer. More...
ssp_err_t(*	volumeSet)(sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume) Set software volume control. Software volume control is applied globally to all streams on the hardware. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Store version information in provided pointer. More...

Detailed Description

Audio playback API structure. Audio playback implementations use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_audio_playback_api_t::close) (sf_audio_playback_ctrl_t *const p_ctrl)

The close API handles cleans up internal driver data.

Implemented as

- SF_AUDIO_PLAYBACK_Close()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.

◆ open

ssp_err_t(* sf_audio_playback_api_t::open) (sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)

Configure the audio framework by creating a thread for audio playback and configuring HAL layer drivers used. This function must be called before any other audio functions.

Implemented as

- SF_AUDIO_PLAYBACK_Open()

Parameters

[in,out]	p_ctrl	Pointer to a device structure allocated by user. The device control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ pause

ssp_err_t(* sf_audio_playback_api_t::pause) (sf_audio_playback_ctrl_t *const p_ctrl)

Pause audio playback. This stops the peripheral that triggers the DMA/DTC transfer and posts a flag to notify SF_AUDIO_PLAYBACK_Start() to pause any playback in progress.

Implemented as

- SF_AUDIO_PLAYBACK_Pause()

Precondition

Call SF_AUDIO_PLAYBACK_Start() before using this function. Calling SF_AUDIO_PLAYBACK_Pause() before SF_AUDIO_PLAYBACK_Start() has no effect and does not return an error code.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.

◆ resume

ssp_err_t(* sf_audio_playback_api_t::resume) (sf_audio_playback_ctrl_t *const p_ctrl)

Resume audio playback. Posts a flag to notify SF_AUDIO_PLAYBACK_Start() to restart the peripheral that triggers the DMA/DTC transfer.

Implemented as

- SF_AUDIO_PLAYBACK_Resume()

Precondition

Call SF_AUDIO_PLAYBACK_Pause() before using this function. Calling SF_AUDIO_PLAYBACK_Resume() before SF_AUDIO_PLAYBACK_Pause() has no effect and does not return an error code.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.

◆ start

ssp_err_t(* sf_audio_playback_api_t::start) (sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)

Play audio. Currently only 16-bit mono PCM buffers are supported.

Implemented as

- SF_AUDIO_PLAYBACK_Start()

Precondition

Call SF_MESSAGE_Open to configure the messaging framework control block and queues with the parameters specified in sf_audio_playback_cfg_t::p_message and sf_audio_playback_cfg_t::p_queue.

Parameters

[in,out]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.
[in]	p_data	Pointer to data, description, timeout values, and synchronization options.
[in]	timeout	ThreadX timeout, represents the maximum amount of time to wait to post to the audio queue. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout values from 0x00000001 through 0xFFFFFFFE in ThreadX tick counts.

◆ stop

ssp_err_t(* sf_audio_playback_api_t::stop) (sf_audio_playback_ctrl_t *const p_ctrl)

Stop audio playback. Causes SF_AUDIO_PLAYBACK_Start() halt playback and return.

Implemented as

- SF_AUDIO_PLAYBACK_Stop()

Precondition

Call SF_AUDIO_PLAYBACK_Start() before using this function. Calling SF_AUDIO_PLAYBACK_Stop() before SF_AUDIO_PLAYBACK_Start() has no effect and does not return an error code.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.

◆ versionGet

ssp_err_t(* sf_audio_playback_api_t::versionGet) (ssp_version_t *const p_version)

Store version information in provided pointer.

Implemented as

- SF_AUDIO_PLAYBACK_VersionGet()

Parameters

[in]	p_version	Pointer to device control block initialized in Open call for UART driver.

◆ volumeSet

ssp_err_t(* sf_audio_playback_api_t::volumeSet) (sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume)

Set software volume control. Software volume control is applied globally to all streams on the hardware.

Implemented as

- SF_AUDIO_PLAYBACK_VolumeSet()

Warning

Software volume control reduces resolution and may require extra memory and processing bandwidth.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for audio driver.
[in]	volume	Volume level requested. Valid range is from 0 (muted, which will stop playback) to 255 (maximum volume, default on open).

The documentation for this struct was generated from the following file:

- sf_audio_playback_api.h

sf_audio_playback_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Framework Interface

#include <sf_audio_playback_api.h>

Data Fields

sf_audio_playback_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_audio_playback_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.

sf_audio_playback_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_audio_playback_api.h

5.1.2.3 Audio Playback Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Audio Playback Framework Interface. More...

Data Structures

struct	sf_audio_playback_data_type_t
struct	sf_audio_playback_hw_callback_args_t
struct	sf_audio_playback_hw_cfg_t
struct	sf_audio_playback_hw_api_t
struct	sf_audio_playback_hw_instance_t

Typedefs

typedef void	sf_audio_playback_hw_ctrl_t

Enumerations

enum	sf_audio_playback_hw_event_t { SF_AUDIO_PLAYBACK_HW_EVENT_PLAYBACK_COMPLETE, SF_AUDIO_PLAYBACK_HW_EVENT_ERROR }

Detailed Description

RTOS-integrated Audio Playback Framework Interface.

Summary

Audio playback driver to play buffers of audio data.

Implemented by: DAC Audio Playback Framework

Audio Framework Interface description: Audio Playback Framework

Typedef Documentation

◆ sf_audio_playback_hw_ctrl_t

typedef void sf_audio_playback_hw_ctrl_t
Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. **Implemented as** ◦ sf_audio_playback_hw_dac_instance_ctrl_t ◦ sf_audio_playback_hw_i2s_instance_ctrl_t

Enumeration Type Documentation

◆ sf_audio_playback_hw_event_t

enum sf_audio_playback_hw_event_t	
Callback event types.	
Enumerator	
SF_AUDIO_PLAYBACK_HW_EVENT_PLAYBACK_COMPLETE	Audio playback complete event.
SF_AUDIO_PLAYBACK_HW_EVENT_ERROR	Audio playback error event.

sf_audio_playback_data_type_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Playback Framework Interface

#include <sf_audio_playback_hw_api.h>

Data Fields

uint8_t	scale_bits_max Maximum data resolution in bits.

bool	is_signed Set to 1 for signed samples, or 0 for unsigned samples.

Detailed Description

Audio data type.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_api.h

sf_audio_playback_hw_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Playback Framework Interface

#include <sf_audio_playback_hw_api.h>

Data Fields

void *	p_context
sf_audio_playback_hw_event_t	event Event that triggered the callback.

Detailed Description

Callback function parameter data

Field Documentation

◆ p_context

void* sf_audio_playback_hw_callback_args_t::p_context
Placeholder for user data. Set in sf_audio_playback_hw_api_t::open function in sf_audio_playback_hw_cfg_t.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_api.h

sf_audio_playback_hw_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Playback Framework Interface

#include <sf_audio_playback_hw_api.h>

Data Fields

void(*	p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
void *	p_context
void const *	p_extend Hardware dependent configuration.

Detailed Description

Audio playback driver configuration.

Field Documentation

◆ p_callback

void(* sf_audio_playback_hw_cfg_t::p_callback) (sf_audio_playback_hw_callback_args_t *p_args)
Callback called when play is complete. Set to NULL for no callback.

◆ p_context

void* sf_audio_playback_hw_cfg_t::p_context
Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_api.h

sf_audio_playback_hw_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Playback Framework Interface

#include <sf_audio_playback_hw_api.h>

Data Fields

ssp_err_t(*	open)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)
ssp_err_t(*	start)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
ssp_err_t(*	stop)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
ssp_err_t(*	play)(sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)
ssp_err_t(*	dataTypeGet)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)
ssp_err_t(*	close)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Audio playback API definition.

Field Documentation

◆ close

ssp_err_t(* sf_audio_playback_hw_api_t::close) (sf_audio_playback_hw_ctrl_t *const p_ctrl)

Close the audio driver.

Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Close()

Parameters

[in]	p_ctrl	Pointer to control block initialized in sf_audio_playback_hw_api_t::open.

◆ dataTypeGet

ssp_err_t(* sf_audio_playback_hw_api_t::dataTypeGet) (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)

Stores expected data type in provided pointer p_data_type.

Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet()

Parameters

[in]	p_ctrl	Pointer to control block initialized in sf_audio_playback_hw_api_t::open.
[out]	p_data_type	Pointer to audio sample data type required by hardware.

◆ open

ssp_err_t(* sf_audio_playback_hw_api_t::open) (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)

Open a device channel for read/write and control.

Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Open()

Parameters

[in,out]	p_ctrl	Pointer to memory allocated for control block.
[in]	p_cfg	Pointer to the hardware configurations.

◆ play

ssp_err_t(* sf_audio_playback_hw_api_t::play) (sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)

Play audio buffer.

Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Play()

Parameters

[in]	p_ctrl	Pointer to control block initialized in sf_audio_playback_hw_api_t::open.
[in]	p_buffer	Pointer to buffer with PCM samples to play. Data must be scaled for audio playback hardware.
[in]	length	Length of data in p_buffer.

◆ start

ssp_err_t(* sf_audio_playback_hw_api_t::start) (sf_audio_playback_hw_ctrl_t *const p_ctrl)

Start audio playback hardware.

Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Start()

Parameters

[in]	p_ctrl	Pointer to control block initialized in sf_audio_playback_hw_api_t::open.

◆ stop

ssp_err_t(* sf_audio_playback_hw_api_t::stop) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
Stop audio playback hardware. **Implemented as** ◦ SF_AUDIO_PLAYBACK_HW_DAC_Stop() **Parameters** [in] p_ctrl — Pointer to control block initialized in sf_audio_playback_hw_api_t::open.

◆ versionGet

ssp_err_t(* sf_audio_playback_hw_api_t::versionGet) (ssp_version_t *const p_version)
Return the version of the driver. **Implemented as** ◦ SF_AUDIO_PLAYBACK_HW_DAC_VersionGet() **Parameters** [out] p_version — Pointer to variable that will be populated with version information.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_api.h

sf_audio_playback_hw_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Playback Framework Interface

#include <sf_audio_playback_hw_api.h>

Data Fields

sf_audio_playback_hw_ctrl_t	p_ctrl

*	Pointer to the control structure for this instance.
sf_audio_playback_hw_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_audio_playback_hw_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_api.h

5.1.2.4 Audio Recording Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Audio Recording Framework Interface. More...

Data Structures

struct	sf_audio_record_cfg_t
struct	sf_audio_record_api_t
struct	sf_audio_record_instance_t

Macros

#define	SF_AUDIO_RECORD_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_audio_record_ctrl_t

Enumerations

enum	sf_audio_record_channel_t { SF_AUDIO_RECORD_CHANNEL_MONO, SF_AUDIO_RECORD_CHANNEL_STEREO }
enum	sf_audio_record_data_size_t { SF_AUDIO_RECORD_DATA_SIZE_8BIT = 1, SF_AUDIO_RECORD_DATA_SIZE_16BIT = 2 }
enum	sf_audio_record_event_t { SF_AUDIO_RECORD_EVENT_NEW_DATA }

Detailed Description

RTOS-integrated Audio Recording Framework Interface.

Summary

The Audio Record Interface is a ThreadX-aware Interface for Audio Recording. The Interface is implemented by the ADC Audio recording Framework using the ADC periodic Framework driver for recording. The interface is implemented by the I2S Audio recording Framework using the I2S driver for recording.

Interfaces used:

- ADC periodic Framework
- I2S Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Macro Definition Documentation

◆ SF_AUDIO_RECORD_API_VERSION_MAJOR

#define SF_AUDIO_RECORD_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_audio_record_ctrl_t

typedef void sf_audio_record_ctrl_t
Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. **Implemented as** ◦ sf_audio_record_adc_instance_ctrl_t ◦ sf_audio_record_i2s_instance_ctrl_t

Enumeration Type Documentation

◆ sf_audio_record_channel_t

enum sf_audio_record_channel_t	
Definition of audio recording mode.	
Enumerator	
SF_AUDIO_RECORD_CHANNEL_MONO	Support Mono Channel.
SF_AUDIO_RECORD_CHANNEL_STEREO	Support Stereo Channel.

◆ sf_audio_record_data_size_t

enum sf_audio_record_data_size_t	
Definition of audio recording data sample size.	
Enumerator	
SF_AUDIO_RECORD_DATA_SIZE_8BIT	data width in the sample is 8bit
SF_AUDIO_RECORD_DATA_SIZE_16BIT	data width in the sample is 16bit

◆ sf_audio_record_event_t

enum sf_audio_record_event_t	
Definition of events for audio recording.	
Enumerator	
SF_AUDIO_RECORD_EVENT_NEW_DATA	New data is available in the buffer.

sf_audio_record_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Recording Framework Interface

#include <sf_audio_record_api.h>

Data Fields

sf_audio_record_data_size_t capture_data_size

	Size of data in the sample 8 or 16 bit.
uint32_t	sampling_rate_hz Sampling rate for audio capture.
void *	p_capture_data_buffer
uint32_t	capture_data_buffer_size
uint32_t	sample_count
void(*	p_callback)(sf_audio_record_callback_args_t *p_args) Callback function.
void const *	p_context Placeholder for user data.
void const *	p_extend

Detailed Description

Configuration for audio recording framework

Field Documentation

◆ capture_data_buffer_size

uint32_t sf_audio_record_cfg_t::capture_data_buffer_size
total size of buffer configured by user to store samples

◆ p_capture_data_buffer

void* sf_audio_record_cfg_t::p_capture_data_buffer
Pointer to the buffer that will store the samples

◆ p_extend

void const* sf_audio_record_cfg_t::p_extend
Extension parameter for hardware specific settings.

◆ sample_count

uint32_t sf_audio_record_cfg_t::sample_count
Samples per channel to be buffered before notifying the user via callback

---

The documentation for this struct was generated from the following file:

- sf_audio_record_api.h

sf_audio_record_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Recording Framework Interface

#include <sf_audio_record_api.h>

Data Fields

ssp_err_t(*	open)(sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_cfg_t const *const p_cfg) Initializes audio recording framework. More...
ssp_err_t(*	start)(sf_audio_record_ctrl_t *const p_ctrl) Starts audio recording. More...
ssp_err_t(*	stop)(sf_audio_record_ctrl_t *const p_ctrl) Stops audio recording. More...
ssp_err_t(*	infoGet)(sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_info_t *p_info) Gets channel information(Mono/Stereo). More...
ssp_err_t(*	close)(sf_audio_record_ctrl_t *const p_ctrl) Releases channel mutex and closes channel at HAL layer. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

	Gets version and stores it in provided pointer p_version. More...

Detailed Description

Framework Audio Recording API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_audio_record_api_t::close) (sf_audio_record_ctrl_t *const p_ctrl)

Releases channel mutex and closes channel at HAL layer.

Implemented as

- SF_AUDIO_RECORD_ADC_Close()
- SF_AUDIO_RECORD_I2S_Close()

Parameters

[in]	p_ctrl	Pointer to control block.

◆ infoGet

ssp_err_t(* sf_audio_record_api_t::infoGet) (sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_info_t *p_info)

Gets channel information(Mono/Stereo).

Implemented as

- SF_AUDIO_RECORD_ADC_InfoGet()
- SF_AUDIO_RECORD_I2S_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control block.
[out]	p_info	Pointer to information block.

◆ open

ssp_err_t(* sf_audio_record_api_t::open) (sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_cfg_t const *const p_cfg)

Initializes audio recording framework.

Implemented as

- SF_AUDIO_RECORD_ADC_Open()
- SF_AUDIO_RECORD_I2S_Open()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. Elements initialized here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ start

ssp_err_t(* sf_audio_record_api_t::start) (sf_audio_record_ctrl_t *const p_ctrl)

Starts audio recording.

Implemented as

- SF_AUDIO_RECORD_ADC_Start()
- SF_AUDIO_RECORD_I2S_Start()

Parameters

[in]	p_ctrl	Pointer to control block set

◆ stop

ssp_err_t(* sf_audio_record_api_t::stop) (sf_audio_record_ctrl_t *const p_ctrl)

Stops audio recording.

Implemented as

- SF_AUDIO_RECORD_ADC_Stop()
- SF_AUDIO_RECORD_I2S_Stop()

Parameters

[in]	p_ctrl	Pointer to control block.

◆ versionGet

ssp_err_t(* sf_audio_record_api_t::versionGet) (ssp_version_t *const p_version)
Gets version and stores it in provided pointer p_version. **Implemented as** ◦ SF_AUDIO_RECORD_ADC_VersionGet() ◦ SF_AUDIO_RECORD_I2S_VersionGet() **Parameters** [out] p_version Code and API version used.

The documentation for this struct was generated from the following file:

- sf_audio_record_api.h

sf_audio_record_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Audio Recording Framework Interface

#include <sf_audio_record_api.h>

Data Fields

sf_audio_record_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_audio_record_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_audio_record_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_audio_record_api.h

5.1.2.5 SF BLE Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Framework Interface. More...

Data Structures

struct	sf_ble_addr_t
struct	sf_ble_adv_data_t
struct	sf_ble_event_info_t
struct	sf_ble_sec_info_t
struct	sf_ble_provisioning_t
struct	sf_ble_scan_info_t
struct	sf_ble_bonding_start_t
struct	sf_ble_bonding_response_t
struct	sf_ble_sm_tk_info_t
struct	sf_ble_sm_tk_ind_t
struct	sf_ble_addr_verify_ind_t
struct	sf_ble_sm_key_ind_t
struct	sf_ble_sm_enc_info_t
struct	sf_ble_sec_enc_start_ind_t
struct	sf_ble_scan_response_data_t
struct	sf_ble_adv_info_t

struct	sf_ble_scan_t
struct	sf_ble_connection_t
struct	sf_ble_disconnect_t
struct	sf_ble_chipset_info_t
struct	sf_ble_info_t
struct	sf_ble_bonding_req_ind_t
struct	sf_ble_connect_info_t
struct	sf_ble_set_tx_pwr_info_t
struct	sf_ble_uuid_t
struct	sf_ble_svc_attribute_t
struct	sf_ble_char_attribute_t
struct	sf_ble_gatt_attr_event_t
struct	sf_ble_service_discovery_rsp_t
struct	sf_ble_service_discovery_req_t
struct	sf_ble_char_discovery_req_t
struct	sf_ble_char_discovery_rsp_t
struct	sf_ble_char_desc_discovery_rsp_t
struct	sf_ble_char_multiple_read_req_t
struct	sf_ble_char_read_req_t
struct	sf_ble_char_multiple_read_rsp_t
struct	sf_ble_char_read_by_uuid_rsp_t
struct	sf_ble_char_read_by_handle_rsp_t
union	sf_ble_char_read_rsp_t
struct	sf_ble_char_write_req_t

struct	sf_ble_gatt_notif_ind_event_data_t
struct	sf_ble_write_cmd_event_data_t
struct	sf_ble_ctrl_t
struct	sf_ble_cfg_t
struct	sf_ble_api_t
struct	sf_ble_instance_t

Macros

#define	SF_BLE_API_VERSION_MAJOR (2U)
#define	SF_BLE_API_VERSION_MINOR (0U)
#define	SF_BLE_MAX_BLE_ADV_DATA_LEN (100U) BLE advertising data length.
#define	SF_BLE_MAX_NAME_LEN (66U) Maximum length of name for BLE device.
#define	SF_BLE_MAX_GAP_NAME_LEN (23U) Maximum length of GAP name.
#define	SF_BLE_ADDR_LEN (6U) BLE address length.
#define	SF_BLE_SEC_KEY_LEN (0x10U) BLE Security Key length.
#define	SF_BLE_128BITS_UUID_LENGTH (16U) 128 bit UUID length
#define	SF_BLE_RAND_NUM_LENGTH (8U) Rand Number length.

#define	SF_BLE_TRUE (1U) Boolean True Condition.
#define	SF_BLE_FALSE (0U) Boolean False Condition.
#define	SF_BLE_MAX_MULTI_CHAR_READ_CNT (4U)
#define	SF_BLE_MAX_CHAR_UUID_READ_CNT (10U)
#define	SF_BLE_GATT_LEN_UNDEF (0xFF)
#define	SF_BLE_ADV_DATA_LEN (0x1F)
#define	SF_BLE_GATT_PRI_SERVICE (0x2800U)
#define	SF_BLE_GATT_INCLUDE_SERVICE (0x2802U)
#define	SF_BLE_GATT_CHAR_DECLARE (0x2803U)
#define	SF_BLE_TX_POWER_CONNECTION_HANDLE (0xFFFFU)

Typedefs

typedef void(*	sf_ble_callback_t) (sf_ble_event_info_t *ev)
typedef uint16_t	sf_ble_conn_handle_t

Enumerations

enum	sf_ble_event_t {
	SF_BLE_EVENT_NONE, SF_BLE_EVENT_BONDING_INDICATION,
	SF_BLE_EVENT_CONNECTION_COMP,
	SF_BLE_EVENT_DISCONNECT_COMP,
	SF_BLE_EVENT_SM_TK_REQ_IND, SF_BLE_EVENT_SM_KEY_IND,
	SF_BLE_EVENT_SM_CHK_BD_ADDR_REQ,
	SF_BLE_EVENT_SM_ENC_START_IND,
	SF_BLE_EVENT_GATT_NOTIFICATION,
	SF_BLE_EVENT_GATT_INDICATION,
	SF_BLE_EVENT_GATT_WRITE_CMD_INDICATION,
	SF_BLE_EVENT_GATT_RESPONSE_TIMEOUT,
	SF_BLE_EVENT_GATT_ATTR_WRITE_REQ,
	SF_BLE_EVENT_GATT_ATTR_READ_REQ,
	SF_BLE_EVENT_GATT_ATTR_WRITE_COMPLETE,
	SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_REQ,
	SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_COMPLETE
	}

enum	sf_ble_gap_role_t { SF_BLE_GAP_ROLE_MASTER,

	SF_BLE_GAP_ROLE_SLAVE, SF_BLE_GAP_ROLE_OBSERVER, SF_BLE_GAP_ROLE_BROADCASTER }
enum	sf_ble_addr_type_t { SF_BLE_ADDR_TYPE_PUBLIC, SF_BLE_ADDR_TYPE_RANDOM }
enum	sf_ble_scan_mode_t { SF_BLE_SCAN_MODE_PASSIVE, SF_BLE_SCAN_MODE_ACTIVE }
enum	sf_ble_bonding_mode_t { SF_BLE_BONDING_MODE_NONBONDABLE, SF_BLE_BONDING_MODE_BONDABLE }
enum	sf_ble_sec_mode_t { SF_BLE_SEC_MODE1_LVL1_NO_SEC, SF_BLE_SEC_MODE1_LVL2_NOAUTH_PAIR_ENC, SF_BLE_SEC_MODE1_LVL3_AUTH_PAIR_ENC, SF_BLE_SEC_MODE1_LVL4_AUTHLE_PAIR_ENC, SF_BLE_SEC_MODE2_LVL1_NOAUTH_DATA_SIGNED, SF_BLE_SEC_MODE2_LVL2_AUTH_DATA_SIGNED }
enum	sf_ble_disc_type_t { SF_BLE_DISC_TYPE_NON_DISCOVERABLE = 0x01, SF_BLE_DISC_TYPE_GENERAL_DISCOVERABLE = 0x02, SF_BLE_DISC_TYPE_LIMITED_DISCOVERABLE = 0x04 }
enum	sf_ble_conn_type_t { SF_BLE_CONN_TYPE_NON_CONNECTABLE, SF_BLE_CONN_TYPE_UNDIRECTED_CONNECTABLE, SF_BLE_CONN_TYPE_DIRECTED_CONNECTABLE }
enum	sf_ble_adv_type_t { SF_BLE_ADV_TYPE_CONN_UNDIR, SF_BLE_ADV_TYPE_CONN_DIR_HIGH_DUTY, SF_BLE_ADV_TYPE_DISC_UNDIR, SF_BLE_ADV_TYPE_NONCONN_UNDIR, SF_BLE_ADV_TYPE_CONN_DIR_LOW_DUTY }
enum	sf_ble_adv_filt_type_t { SF_BLE_ADV_FILT_TYPE_ALLOW_ALL, SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_WLIST_CON_ANY, SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_ANY_CON_WLIST, SF_BLE_ADV_FILT_TYPE_ALLOW_WLIST_ONLY }
enum	sf_ble_duplicate_filter_t { SF_BLE_DUPLICATE_FILTER_ENABLE, SF_BLE_DUPLICATE_FILTER_DISABLE }
enum	sf_ble_init_filt_type_t { SF_BLE_INIT_FILT_TYPE_IGNORE_WLIST, SF_BLE_INIT_FILT_TYPE_USE_WLIST }
enum	sf_ble_adv_chnl_map_t { SF_BLE_ADV_CHNL_37, SF_BLE_ADV_CHNL_38, SF_BLE_ADV_CHNL_39, SF_BLE_ADV_CHNL_ALL_CHANNELS }

enum	sf_ble_iocap_t { SF_BLE_IO_CAP_DISPLAY_ONLY, SF_BLE_IO_CAP_DISPLAY_YES_NO, SF_BLE_IO_CAP_KB_ONLY, SF_BLE_IO_CAP_NO_INPUT_NO_OUTPUT, SF_BLE_IO_CAP_KB_DISPLAY }
enum	sf_ble_auth_type_t { SF_BLE_AUTH_TYPE_NONE, SF_BLE_AUTH_TYPE_UNAUTHENTICATED_NO_MITM, SF_BLE_AUTH_TYPE_AUTHENTICATED_MITM }
enum	sf_ble_disconnect_reason_t { SF_BLE_DISCONNECT_REASON_LOCAL_HOST, SF_BLE_DISCONNECT_REASON_REMOTE_USER, SF_BLE_DISCONNECT_REASON_ERR }
enum	sf_ble_key_dist_t { SF_BLE_KEY_DIST_NONE = 0x00, SF_BLE_KEY_DIST_ENCKEY = 0x01, SF_BLE_KEY_DIST_IDKEY = 0x02, SF_BLE_KEY_DIST_SIGNKEY = 0x04 }
enum	sf_ble_tx_pwr_state_t { SF_BLE_TX_PWR_STATE_NORMAL, SF_BLE_TX_PWR_STATE_ADAPT_NEAR, SF_BLE_TX_PWR_STATE_ADAPT_MIDDLE, SF_BLE_TX_PWR_STATE_ADAPT_FAR }
enum	sf_ble_char_property_t { SF_BLE_CHAR_PROPERTY_BCAST = 0x01U, SF_BLE_CHAR_PROPERTY_RD = 0x02U, SF_BLE_CHAR_PROPERTY_WR_NO_RESP = 0x04U, SF_BLE_CHAR_PROPERTY_WR = 0x08U, SF_BLE_CHAR_PROPERTY_NTF = 0x10U, SF_BLE_CHAR_PROPERTY_IND = 0x20U, SF_BLE_CHAR_PROPERTY_AUTH = 0x40U, SF_BLE_CHAR_PROPERTY_EXT_PROP = 0x80U }
enum	sf_ble_service_discovery_t { SF_BLE_SERVICE_DISCOVERY_PRIMARY_ALL = 0x1, SF_BLE_SERVICE_DISCOVERY_PRIMARY_UUID = 0x10, SF_BLE_SERVICE_DISCOVERY_INCLUDED = 0x100 }
enum	sf_ble_char_discovery_t { SF_BLE_CHAR_DISCOVERY_ALL = 0x1, SF_BLE_CHAR_DISCOVERY_UUID = 0x10 }
enum	sf_ble_uuid_length_t { SF_BLE_UUID_LENGTH_16BITS = 2, SF_BLE_UUID_LENGTH_32BITS = 4, SF_BLE_UUID_LENGTH_128BITS = 16 }
enum	sf_ble_char_read_t { SF_BLE_CHAR_READ_BY_HANDLE = 1, SF_BLE_CHAR_READ_BY_UUID = 2, SF_BLE_CHAR_READ_LONG_BY_HANDLE = 3, SF_BLE_CHAR_DESC_READ_BY_HANDLE = 4,

	SF_BLE_CHAR_DESC_READ_LONG_BY_HANDLE = 5, SF_BLE_CHAR_READ_MULTIPLE_BY_HANDLES = 6 }

enum	sf_ble_char_write_t {
	SF_BLE_CHAR_WRITE_NO_RESPONSE = 1, SF_BLE_CHAR_WRITE = 2, SF_BLE_CHAR_WRITE_LONG = 3, SF_BLE_CHAR_DESC_WRITE = 4,
	SF_BLE_CHAR_DESC_WRITE_LONG = 5
	}

enum	sf_ble_execute_write_t { SF_BLE_EXECUTE_WRITE_FALSE = 0, SF_BLE_EXECUTE_WRITE_TRUE = 1 }
enum	sf_ble_write_response_t { SF_BLE_WRITE_RESPONSE_NOT_REQUIRED = 0, SF_BLE_WRITE_RESPONSE_REQUIRED = 1 }

enum	sf_ble_attribute_error_code_t {
	SF_BLE_ATTRIBUTE_ERR_SUCCESS = 0x00,
	SF_BLE_ATTRIBUTE_ERR_INVALID_HANDLE = 0x01,
	SF_BLE_ATTRIBUTE_ERR_WRITE_NOT_PERMITTED = 0x03,
	SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHEN = 0x05,
	SF_BLE_ATTRIBUTE_ERR_REQUEST_NOT_SUPPORTED = 0x06,
	SF_BLE_ATTRIBUTE_ERR_INVALID_OFFSET = 0x07,
	SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHOR = 0x08,
	SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_FOUND = 0x0a,
	SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_LONG = 0x0b,
	SF_BLE_ATTRIBUTE_ERR_INVALID_ATTRIBUTE_VAL_LEN = 0x0d,
	SF_BLE_ATTRIBUTE_ERR_INSUFF_ENC = 0x0f,
	SF_BLE_ATTRIBUTE_ERR_OUT_OF_RANGE = 0xFF
	}

enum	sf_ble_char_attr_permissions_t {
	SF_BLE_ATT_PERMISSIONS_ALLACCESS = 0x00u,
	SF_BLE_ATT_PERMISSIONS_READ_AUTHENTICATION = 0x01u,
	SF_BLE_ATT_PERMISSIONS_READ_ENCRYPTION = 0x02u,
	SF_BLE_ATT_PERMISSIONS_READ_AUTHORIZATION = 0x04u,
	SF_BLE_ATT_PERMISSIONS_READ_FORBIDDEN = 0x08u,
	SF_BLE_ATT_PERMISSIONS_WRITE_AUTHENTICATION = 0x10u,
	SF_BLE_ATT_PERMISSIONS_WRITE_ENCRYPTION = 0x20u,
	SF_BLE_ATT_PERMISSIONS_WRITE_AUTHORIZATION = 0x40u,
	SF_BLE_ATT_PERMISSIONS_WRITE_FORBIDDEN = 0x80u,
	SF_BLE_ATT_PERMISSIONS_NOACCESS = 0x88u
	}

enum	sf_ble_adv_event_type_t {
	SF_BLE_ADV_EVENT_TYPE_CONN_UND_ADV,
	SF_BLE_ADV_EVENT_TYPE_CONN_DIR_ADV,
	SF_BLE_ADV_EVENT_TYPE_SCANNABLE_UND_ADV,
	SF_BLE_ADV_EVENT_TYPE_NON_CONN_UND_ADV,
	SF_BLE_ADV_EVENT_TYPE_SCAN_RESPONSE
	}

Detailed Description

RTOS-integrated SF BLE Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Framework.

Macro Definition Documentation

◆ SF_BLE_ADV_DATA_LEN

#define SF_BLE_ADV_DATA_LEN (0x1F)
BLE Advertising Data Length

◆ SF_BLE_API_VERSION_MAJOR

#define SF_BLE_API_VERSION_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_BLE_API_VERSION_MINOR

#define SF_BLE_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

◆ SF_BLE_GATT_CHAR_DECLARE

#define SF_BLE_GATT_CHAR_DECLARE (0x2803U)
BLE GATT Declare Characteristics

◆ SF_BLE_GATT_INCLUDE_SERVICE

#define SF_BLE_GATT_INCLUDE_SERVICE (0x2802U)
BLE GATT Service type Included

◆ SF_BLE_GATT_LEN_UNDEF

#define SF_BLE_GATT_LEN_UNDEF (0xFF)
Variable length characteristic value which can not be defined

◆ SF_BLE_GATT_PRI_SERVICE

#define SF_BLE_GATT_PRI_SERVICE (0x2800U)
BLE GATT Service type Primary

◆ SF_BLE_MAX_CHAR_UUID_READ_CNT

#define SF_BLE_MAX_CHAR_UUID_READ_CNT (10U)
Maximum characteristics count in result of characteristic read by UUID

◆ SF_BLE_MAX_MULTI_CHAR_READ_CNT

#define SF_BLE_MAX_MULTI_CHAR_READ_CNT (4U)
Maximum characteristics count for multiple read

◆ SF_BLE_TX_POWER_CONNECTION_HANDLE

#define SF_BLE_TX_POWER_CONNECTION_HANDLE (0xFFFFU)
BLE connection handle value when TX power is set

Typedef Documentation

◆ sf_ble_callback_t

typedef void(* sf_ble_callback_t) (sf_ble_event_info_t *ev)
BLE Callback Type

◆ sf_ble_conn_handle_t

typedef uint16_t sf_ble_conn_handle_t
BLE Connection Handle

Enumeration Type Documentation

◆ sf_ble_addr_type_t

enum sf_ble_addr_type_t	
BLE address type	
Enumerator	
SF_BLE_ADDR_TYPE_PUBLIC	BLE address type public.
SF_BLE_ADDR_TYPE_RANDOM	BLE address type random.

◆ sf_ble_adv_chnl_map_t

enum sf_ble_adv_chnl_map_t	
BLE advertisement channel map	
Enumerator	
SF_BLE_ADV_CHNL_37	Enable channel 37 use.
SF_BLE_ADV_CHNL_38	Enable channel 38 use.
SF_BLE_ADV_CHNL_39	Enable channel 39 use.
SF_BLE_ADV_CHNL_ALL_CHANNELS	all channels(37, 38 and 39) enabled

◆ sf_ble_adv_event_type_t

enum sf_ble_adv_event_type_t	
BLE advertising event type	
Enumerator	
SF_BLE_ADV_EVENT_TYPE_CONN_UND_ADV	Connectable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_CONN_DIR_ADV	Connectable directed advertising.
SF_BLE_ADV_EVENT_TYPE_SCANNABLE_UND_ADV	Scannable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_NON_CONN_UND_ADV	Non connectable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_SCAN_RESPONSE	Scan response.

◆ sf_ble_adv_filt_type_t

enum sf_ble_adv_filt_type_t	
BLE advertisement filter type	
Enumerator	
SF_BLE_ADV_FILT_TYPE_ALLOW_ALL	Filter type allow all.
SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_WLIST_CON_ANY	Filter type scan only whitelist entries, connect to any.
SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_ANY_CON_WLIST	Filter type scan any, connect to whitelist entries only.
SF_BLE_ADV_FILT_TYPE_ALLOW_WLIST_ONLY	Filter type allow whitelist only for both scan and connection.

◆ sf_ble_adv_type_t

enum sf_ble_adv_type_t	
BLE advertisement type	
Enumerator	
SF_BLE_ADV_TYPE_CONN_UNDIR	Connectable Undirected advertising.
SF_BLE_ADV_TYPE_CONN_DIR_HIGH_DUTY	Connectable high duty cycle directed advertising.
SF_BLE_ADV_TYPE_DISC_UNDIR	Discoverable undirected advertising.
SF_BLE_ADV_TYPE_NONCONN_UNDIR	Non-connectable undirected advertising.
SF_BLE_ADV_TYPE_CONN_DIR_LOW_DUTY	Connectable low duty cycle directed advertising.

◆ sf_ble_attribute_error_code_t

enum sf_ble_attribute_error_code_t	
BLE Attribute write response code	
Enumerator	
SF_BLE_ATTRIBUTE_ERR_SUCCESS	Success.
SF_BLE_ATTRIBUTE_ERR_INVALID_HANDLE	Invalid handle.
SF_BLE_ATTRIBUTE_ERR_WRITE_NOT_PERMITTED	Writing is not permitted.
SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHEN	Authentication required for the request.
SF_BLE_ATTRIBUTE_ERR_REQUEST_NOT_SUPPORTED	Unsupported request.
SF_BLE_ATTRIBUTE_ERR_INVALID_OFFSET	Invalid offset.
SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHOR	Authorization required for the request.
SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_FOUND	The attribute could not be found.
SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_LONG	The attribute is not long enough.
SF_BLE_ATTRIBUTE_ERR_INVALID_ATTRIBUTE_VAL_LEN	Invalid attribute value size.
SF_BLE_ATTRIBUTE_ERR_INSUFF_ENC	Encryption required for the request.
SF_BLE_ATTRIBUTE_ERR_OUT_OF_RANGE	Out of Range.

◆ sf_ble_auth_type_t

enum sf_ble_auth_type_t	
BLE security authorization requirement	
Enumerator	
SF_BLE_AUTH_TYPE_NONE	No Security.
SF_BLE_AUTH_TYPE_UNAUTHENTICATED_NO_MITM	No MITM.
SF_BLE_AUTH_TYPE_AUTHENTICATED_MITM	MITM Protected.

◆ sf_ble_bonding_mode_t

enum sf_ble_bonding_mode_t	
BLE bonding mode	
Enumerator	
SF_BLE_BONDING_MODE_NONBONDABLE	BLE bonding not supported.
SF_BLE_BONDING_MODE_BONDABLE	BLE bonding supported.

◆ sf_ble_char_attr_permissions_t

enum sf_ble_char_attr_permissions_t	
BLE GATT Attribute characteristics permission	
Enumerator	
SF_BLE_ATT_PERMISSIONS_ALLACCESS	Full access.
SF_BLE_ATT_PERMISSIONS_READ_AUTHENTICATION	Read with authentication.
SF_BLE_ATT_PERMISSIONS_READ_ENCRYPTION	Read with encryption.
SF_BLE_ATT_PERMISSIONS_READ_AUTHORIZATION	Read with authorization.
SF_BLE_ATT_PERMISSIONS_READ_FORBIDDEN	Read forbidden.
SF_BLE_ATT_PERMISSIONS_WRITE_AUTHENTICATION	Write with authentication.
SF_BLE_ATT_PERMISSIONS_WRITE_ENCRYPTION	Write with encryption.
SF_BLE_ATT_PERMISSIONS_WRITE_AUTHORIZATION	Write with authorization.
SF_BLE_ATT_PERMISSIONS_WRITE_FORBIDDEN	Write forbidden.
SF_BLE_ATT_PERMISSIONS_NOACCESS	No access.

◆ sf_ble_char_discovery_t

enum sf_ble_char_discovery_t	
Characteristics discovery type	
Enumerator	
SF_BLE_CHAR_DISCOVERY_ALL	Discover all characteristics.
SF_BLE_CHAR_DISCOVERY_UUID	Discover characteristics by UUID.

◆ sf_ble_char_property_t

enum sf_ble_char_property_t	
Characteristic property	
Enumerator	
SF_BLE_CHAR_PROPERTY_BCAST	Permits broadcasts of the Characteristic Value.
SF_BLE_CHAR_PROPERTY_RD	Permits reads of the Characteristic Value.
SF_BLE_CHAR_PROPERTY_WR_NO_RESP	Permits writes of the Characteristic Value without response.
SF_BLE_CHAR_PROPERTY_WR	Permits writes of the Characteristic Value with response.
SF_BLE_CHAR_PROPERTY_NTF	Permits notifications of the Characteristic Value without acknowledgment.
SF_BLE_CHAR_PROPERTY_IND	Permits indications of a Characteristic Value with acknowledgment.
SF_BLE_CHAR_PROPERTY_AUTH	Permits signed writes to the Characteristic Value.
SF_BLE_CHAR_PROPERTY_EXT_PROP	Additional characteristic properties.

◆ sf_ble_char_read_t

enum sf_ble_char_read_t	
Characteristics read type	
Enumerator	
SF_BLE_CHAR_READ_BY_HANDLE	Read single characteristic by handle.
SF_BLE_CHAR_READ_BY_UUID	Read single characteristic by UUID.
SF_BLE_CHAR_READ_LONG_BY_HANDLE	Read single long characteristic by handle.
SF_BLE_CHAR_DESC_READ_BY_HANDLE	Read single characteristic descriptor by handle.
SF_BLE_CHAR_DESC_READ_LONG_BY_HANDLE	Read single long characteristic descriptor by handle.
SF_BLE_CHAR_READ_MULTIPLE_BY_HANDLES	Read multiple characteristics by handles.

◆ sf_ble_char_write_t

enum sf_ble_char_write_t	
Characteristics write type	
Enumerator	
SF_BLE_CHAR_WRITE_NO_RESPONSE	Write single characteristic with no response.
SF_BLE_CHAR_WRITE	Write single characteristic with response.
SF_BLE_CHAR_WRITE_LONG	Write single long characteristic with response.
SF_BLE_CHAR_DESC_WRITE	Write single characteristic descriptor with response.
SF_BLE_CHAR_DESC_WRITE_LONG	Write single long characteristic descriptor with response.

◆ sf_ble_conn_type_t

enum sf_ble_conn_type_t	
BLE connection type	
Enumerator	
SF_BLE_CONN_TYPE_NON_CONNECTABLE	Connection type connection not allowed.
SF_BLE_CONN_TYPE_UNDIRECTED_CONNECTABLE	Connection type undirected.
SF_BLE_CONN_TYPE_DIRECTED_CONNECTABLE	Connection type directed.

◆ sf_ble_disc_type_t

enum sf_ble_disc_type_t	
BLE discovery type	
Enumerator	
SF_BLE_DISC_TYPE_NON_DISCOVERABLE	BLE non-discoverable.
SF_BLE_DISC_TYPE_GENERAL_DISCOVERABLE	BLE discoverable.
SF_BLE_DISC_TYPE_LIMITED_DISCOVERABLE	BLE limited discoverable.

◆ sf_ble_disconnect_reason_t

enum sf_ble_disconnect_reason_t	
BLE security authorization requirement	
Enumerator	
SF_BLE_DISCONNECT_REASON_LOCAL_HOST	Disconnected by local machine.
SF_BLE_DISCONNECT_REASON_REMOTE_USER	Disconnected by remote device.
SF_BLE_DISCONNECT_REASON_ERR	Error in Connection so disconnected.

◆ sf_ble_duplicate_filter_t

enum sf_ble_duplicate_filter_t	
BLE Duplicate Filter Configuration	
Enumerator	
SF_BLE_DUPLICATE_FILTER_ENABLE	Enable Duplicate filter in Scan.
SF_BLE_DUPLICATE_FILTER_DISABLE	Disbale Duplicate filter in Scan.

◆ sf_ble_event_t

enum sf_ble_event_t	
BLE events	
Enumerator	
SF_BLE_EVENT_NONE	BLE user event none.
SF_BLE_EVENT_BONDING_INDICATION	BLE user event indicating reception of bonding request.
SF_BLE_EVENT_CONNECTION_COMP	BLE user event indicating connection completion.
SF_BLE_EVENT_DISCONNECT_COMP	BLE user event indicating disconnect.
SF_BLE_EVENT_SM_TK_REQ_IND	BLE user event indicating request for Temporary key.
SF_BLE_EVENT_SM_KEY_IND	BLE user event indicating received key from user.
SF_BLE_EVENT_SM_CHK_BD_ADDR_REQ	BLE user event indicating validation of remote address.
SF_BLE_EVENT_SM_ENC_START_IND	BLE user event indicating encryption started.
SF_BLE_EVENT_GATT_NOTIFICATION	BLE user event for Notification from peer.
SF_BLE_EVENT_GATT_INDICATION	BLE user event for Indication from peer.
SF_BLE_EVENT_GATT_WRITE_CMD_INDICATION	BLE user event for Write command from peer.
SF_BLE_EVENT_GATT_RESPONSE_TIMEOUT	BLE user event for GATT operation timeout.
SF_BLE_EVENT_GATT_ATTR_WRITE_REQ	BLE user event for Remote Write request on

	user Attributes.
SF_BLE_EVENT_GATT_ATTR_READ_REQ	BLE user event for Remote Read request on user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_COMPLETE	BLE user event Indicating Remote Write complete for user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_REQ	BLE user event for Remote CCCD Write request on user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_COMPLETE	BLE user event Indicating Remote CCCD Write complete for user Attributes.

◆ sf_ble_execute_write_t

enum sf_ble_execute_write_t	
Characteristic execute write flag	
Enumerator	
SF_BLE_EXECUTE_WRITE_FALSE	Execute write false i.e. cancel write operations.
SF_BLE_EXECUTE_WRITE_TRUE	Execute write true i.e. execute write operations.

◆ sf_ble_gap_role_t

enum sf_ble_gap_role_t	
GAP role	
Enumerator	
SF_BLE_GAP_ROLE_MASTER	GAP role master/central.
SF_BLE_GAP_ROLE_SLAVE	GAP role slave/peripheral.
SF_BLE_GAP_ROLE_OBSERVER	GAP role Observer.
SF_BLE_GAP_ROLE_BROADCASTER	GAP role Broadcaster.

◆ sf_ble_init_filt_type_t

enum sf_ble_init_filt_type_t	
BLE initial filter type	
Enumerator	
SF_BLE_INIT_FILT_TYPE_IGNORE_WLIST	Ignore whitelist.
SF_BLE_INIT_FILT_TYPE_USE_WLIST	Use whitelist.

◆ sf_ble_iocap_t

enum sf_ble_iocap_t	
BLE System IO Capabilities	
Enumerator	
SF_BLE_IO_CAP_DISPLAY_ONLY	Display Only.
SF_BLE_IO_CAP_DISPLAY_YES_NO	Display Yes No.
SF_BLE_IO_CAP_KB_ONLY	Keyboard Only.
SF_BLE_IO_CAP_NO_INPUT_NO_OUTPUT	No Input No Output.
SF_BLE_IO_CAP_KB_DISPLAY	Keyboard Display.

◆ sf_ble_key_dist_t

enum sf_ble_key_dist_t	
BLE security key distribution type	
Enumerator	
SF_BLE_KEY_DIST_NONE	No Keys to distribute.
SF_BLE_KEY_DIST_ENCKEY	Encryption key in distribution.
SF_BLE_KEY_DIST_IDKEY	IRK (ID key)in distribution.
SF_BLE_KEY_DIST_SIGNKEY	CSRK(Signature key) in distribution.

◆ sf_ble_scan_mode_t

enum sf_ble_scan_mode_t	
BLE scan mode	
Enumerator	
SF_BLE_SCAN_MODE_PASSIVE	BLE scan mode passive.
SF_BLE_SCAN_MODE_ACTIVE	BLE scan mode active.

◆ sf_ble_sec_mode_t

enum sf_ble_sec_mode_t	
BLE security mode	
Enumerator	
SF_BLE_SEC_MODE1_LVL1_NO_SEC	BLE no security.
SF_BLE_SEC_MODE1_LVL2_NOAUTH_PAIR_ENC	BLE no authentication pairing encryption.
SF_BLE_SEC_MODE1_LVL3_AUTH_PAIR_ENC	Authenticated pairing with encryption.
SF_BLE_SEC_MODE1_LVL4_AUTHLE_PAIR_ENC	Authenticated LE Secure Connections pairing with encryption.
SF_BLE_SEC_MODE2_LVL1_NOAUTH_DATA_SIGNED	Unauthenticated pairing with data signing.
SF_BLE_SEC_MODE2_LVL2_AUTH_DATA_SIGNED	Authentication pairing with data signing.

◆ sf_ble_service_discovery_t

enum sf_ble_service_discovery_t	
Service discovery type	
Enumerator	
SF_BLE_SERVICE_DISCOVERY_PRIMARY_ALL	Discover all primary services.
SF_BLE_SERVICE_DISCOVERY_PRIMARY_UUID	Discover primary service by UUID.
SF_BLE_SERVICE_DISCOVERY_INCLUDED	Discover includes services.

◆ sf_ble_tx_pwr_state_t

enum sf_ble_tx_pwr_state_t	
BLE Transmit power state	
Enumerator	
SF_BLE_TX_PWR_STATE_NORMAL	Functionality disabled.
SF_BLE_TX_PWR_STATE_ADAPT_NEAR	RF low-power mode.
SF_BLE_TX_PWR_STATE_ADAPT_MIDDLE	RF normal mode.
SF_BLE_TX_PWR_STATE_ADAPT_FAR	RF high-performance mode.

◆ sf_ble_uuid_length_t

enum sf_ble_uuid_length_t	
UUID length	
Enumerator	
SF_BLE_UUID_LENGTH_16BITS	16 bit UUID
SF_BLE_UUID_LENGTH_32BITS	32 bit UUID
SF_BLE_UUID_LENGTH_128BITS	128 bit UUID

◆ sf_ble_write_response_t

enum sf_ble_write_response_t	
Characteristic execute write flag	
Enumerator	
SF_BLE_WRITE_RESPONSE_NOT_REQUIRED	Write response not required.
SF_BLE_WRITE_RESPONSE_REQUIRED	Write response required.

sf_ble_addr_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	addr [SF_BLE_ADDR_LEN] 6-byte array address value

Detailed Description

BLE address

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_adv_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	data [SF_BLE_ADV_DATA_LEN] Advertising data bytes array.
uint8_t	adv_data_length Advertising data length.

Detailed Description

BLE Advertising data structure

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_event_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

void *	p_data Data for BLE event.
uint16_t	event Event type.

Detailed Description

BLE event info

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sec_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_sec_mode_t	sec_mode security mode
uint8_t	id_key [SF_BLE_SEC_KEY_LEN] GAP Identity Resolving Security key.
uint8_t	csrk_key [SF_BLE_SEC_KEY_LEN] GAP Connection Signature Resolving Security key.

uint8_t	ltk_key [SF_BLE_SEC_KEY_LEN] GAP Connection Long term Security key.
uint8_t	rand_num [SF_BLE_RAND_NUM_LENGTH] GAP Connection Random number for Long term Security key.
uint16_t	ediv GAP Connection Encrypted Diversifier for Long term Security key.

Detailed Description

Security Related Configuration

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_provisioning_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	gap_name [SF_BLE_MAX_GAP_NAME_LEN] GAP name.
uint8_t	bcast_mode broadcast mode
sf_ble_bonding_mode_t	bonding_mode bonding mode
sf_ble_sec_info_t	sec_info

	Security information.
sf_ble_gap_role_t	gap_role GAP role (master/slave)
sf_ble_callback_t	p_ble_callback

Detailed Description

BLE provisioning information

Field Documentation

◆ p_ble_callback

sf_ble_callback_t sf_ble_provisioning_t::p_ble_callback
GAP user event callback that runs in driver thread context. Application should make sure callback logic is as minimal as possible without any blocking calls

---

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_scan_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	total_scan_duration BLE total scan duration in milliseconds.
sf_ble_scan_mode_t	scan_mode BLE scan mode.
sf_ble_disc_type_t	discovery_type

	Specifies discovery type, Used only in active scan.
sf_ble_addr_type_t	address_type BLE address type.
sf_ble_adv_filt_type_t	filt_policy Scan Filter policy.
sf_ble_duplicate_filter_t	duplicate_filt Duplicate filter configuration.

Detailed Description

BLE scan information structure

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_bonding_start_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_iocap_t	iocap IO capabilities.
uint8_t	key_size Encryption key size.
sf_ble_key_dist_t	ikey_dist Initiator key distribution.

sf_ble_key_dist_t	rkey_dist Responder key distribution.

Detailed Description

BLE Bonding Start information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_bonding_response_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	accept accept or reject bonding
sf_ble_iocap_t	io_cap IO capabilities.
uint8_t	max_key_size Max key size.
sf_ble_key_dist_t	ikeys Initiator key distribution.
sf_ble_key_dist_t	rkeys Responder key distribution.

Detailed Description

Bonding Response Parameter

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sm_tk_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
uint8_t	disp_en Whether to enable display.

Detailed Description

Security structures BLE Temporary Key information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sm_tk_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_sm_tk_info_t	tk_info input parameter,information to app about temporary key request

uint8_t	tk_req_status output parameter: request status, application has to set this in callback event whether request is valid or not
uint8_t	tk_key [SF_BLE_SEC_KEY_LEN] application has to set this in callback event temporary key for encryption

Detailed Description

BLE Temporary Key indication

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_addr_verify_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	bd_addr [SF_BLE_ADDR_LEN] input parameter specifying bluetooth address of remote device
sf_ble_addr_type_t	addr_type input parameter specifying Address type of remote BLE device
uint8_t	accept_addr output parameter: application has to set this parameter in callback if it accepts this address

Detailed Description

BLE Bluetooth address verification indication

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sm_key_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	conn_idx connection index
sf_ble_key_dist_t	key_code type of security key
uint16_t	ediv BLE Security EDIV.
uint8_t	rand_num [SF_BLE_RAND_NUM_LENGTH] Random number for security.
uint8_t	ltk_key [SF_BLE_SEC_KEY_LEN] BLE long term security key.

Detailed Description

BLE Received Key information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sm_enc_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_conn_handle_t	conn_idx connection index
uint16_t	ediv BLE Security EDIV.
uint8_t	rand_num [SF_BLE_RAND_NUM_LENGTH] Random number for security.
uint8_t	ltk_key [SF_BLE_SEC_KEY_LEN] BLE long term security key.

Detailed Description

BLE Start Encryption information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_sec_enc_start_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle of the remote device.

uint8_t	status Result of encryption start, 0 = SUCCESS else error.
uint8_t	key_size Key Size.
sf_ble_auth_type_t	auth_type Security properties.
uint8_t	bonding_status Bonding Status, 0 = Unbonded, 1 = Bonded.

Detailed Description

Encryption Start Indication Event for user

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_scan_response_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	scan_response_data [SF_BLE_ADV_DATA_LEN] Advertising data bytes array.
uint8_t	scan_response_data_length Advertising data length.

Detailed Description

Scan Response Data for advertising

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_adv_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_disc_type_t	disc_mode Discovery mode.
sf_ble_conn_type_t	conn_mode Connection mode.
sf_ble_adv_type_t	adv_type Advertisement type.
uint16_t	adv_intv_min Minimum interval for advertising.
uint16_t	adv_intv_max Maximum interval for advertising.
sf_ble_addr_type_t	own_addr_type Own address type.
sf_ble_addr_type_t	direct_addr_type Direct connection address type.
uint8_t	direct_bd_addr [SF_BLE_ADDR_LEN]

	Direct connection BLE address.
sf_ble_adv_chnl_map_t	adv_chnl_map Advertising channel map.
sf_ble_adv_filt_type_t	adv_filt_policy Advertising filter policy.
sf_ble_scan_response_data_t	scan_response_data Scan Response information, will be advertised only for active scan.
sf_ble_adv_data_t	adv_data Advertising data.

Detailed Description

BLE advertisement information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_scan_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_addr_type_t	addr_type Address type of remote BLE device.
uint8_t	bd_addr [SF_BLE_ADDR_LEN] Remote BLE address.

uint16_t	data_len Scan data length.
uint8_t	data [SF_BLE_MAX_BLE_ADV_DATA_LEN] Scan data.
int16_t	rssi RSSI value.
sf_ble_adv_event_type_t	event_type Advertising event type.

Detailed Description

BLE scan information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_connection_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_init_filt_type_t	init_filt_type Connection filter type.
sf_ble_addr_type_t	addr_type BLE address type.
uint8_t	bd_addr [SF_BLE_ADDR_LEN]

	BLE address.

Detailed Description

BLE connection information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_disconnect_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_disconnect_reason_t	reason Disconnection reason.
uint8_t	status Disconnect status if initiated by local host.
sf_ble_conn_handle_t	conhdl Connection handle of the remote device.

Detailed Description

BLE Disconnect Event Information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_chipset_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint32_t	version chipset version
uint8_t	bd_addr [SF_BLE_ADDR_LEN] BLE address.

Detailed Description

BLE chipset information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_chipset_info_t	chipset Chipset information.
uint16_t	rssi RSSI value.

Detailed Description

BLE module information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_bonding_req_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_addr_t	bd_addr BLE address.
uint8_t	index Connection index.
uint8_t	auth_req Authentication request type.
uint8_t	io_cap IO capability.
uint8_t	oob_data_flg Indicating if OOB data is present.
uint8_t	max_enc_size Maximum encryption key size.
uint8_t	ikey_dist Type of key distributed by the initiator.
uint8_t	rkey_dist Type of key distributed by the responder.

Detailed Description

BLE bonding request indication information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_connect_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	status Confirmation status.
uint8_t	reserved Reserved.
sf_ble_conn_handle_t	conhdl Connection handle.
uint8_t	peer_addr_type Peer address type.
sf_ble_addr_t	peer_addr Peer BT address.
uint8_t	reserved2 Reserved.
uint16_t	con_interval Connection interval.

uint16_t	con_latency Connection latency.
uint16_t	sup_to Link supervision time-out.
uint8_t	clk_accuracy Clock accuracy.
uint8_t	reserved3 Reserved.

Detailed Description

BLE connection information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_set_tx_pwr_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

int8_t	power_lvl TX power level in dBm.
sf_ble_tx_pwr_state_t	state Operating state to set the transmit power level.

Detailed Description

BLE set TX power information structure

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_uuid_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_uuid_length_t	uuid_length Service UUID length.
uint16_t	uuid16 16 bit UUID
uint32_t	uuid32 32 bit UUID
uint8_t	uuid128 [SF_BLE_128BITS_UUID_LENGTH] 128 bit UUID

Detailed Description

Union holding either 16-bit/32-bit/128-bit UUID

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_svc_attribute_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	attr_handle Service Handle.
uint16_t	attr_type Attribute type such as SF_BLE_GATT_PRI_SERVICE, SF_BLE_GATT_INCLUDE_SERVICE.
uint16_t	parent_svc_handle parent_svc_handle is service handle of parent service which is already registered More...
uint8_t *	p_attr_value Service UUID Pointer.
sf_ble_uuid_length_t	attr_value_len UUID Length.

Detailed Description

BLE GATT Service Attributes

Field Documentation

◆ parent_svc_handle

uint16_t sf_ble_svc_attribute_t::parent_svc_handle
parent_svc_handle is service handle of parent service which is already registered If service is included service,

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_attribute_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_svc_attribute_t *	p_service Service to which this characteristics belongs.
sf_ble_uuid_t	attr_uuid UUID of characteristics value.
uint16_t	attr_declare_handle Characteristics handle.
uint16_t	attr_declare_type Characteristics declare type SF_BLE_GATT_CHAR_DECLARE.
uint16_t	attr_value_handle Characteristics value handle.
sf_ble_char_attr_permissions_t	attr_perm Characteristics permission.
sf_ble_char_property_t	attr_properties Characteristics properties.
uint8_t *	p_attr_value Characteristics value data.
uint8_t	attr_value_len Characteristics value data length.

Detailed Description

BLE GATT Characteristics Attributes

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_gatt_attr_event_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	attr_handle Attribute handle for which event is generated.
uint16_t	offset The offset at which the client is willing to write.
uint8_t const *	p_value The value at which the client is willing to write.
uint16_t	length The amount of bytes that the client is willing to write as from this offset.
ssp_err_t	response User will update this variable, Pass SSP_SUCCESS if user accepts the remote request.

Detailed Description

BLE GATT Characteristics Remote Event data, passed in provision callback for GATT Custom profile

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_service_discovery_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	service_handle Service handle.
sf_ble_uuid_t	uuid Service UUID.
uint16_t	start_handle Start handle of sub-attributes handle range.
uint16_t	end_handle End handle of sub-attributes handle range.

Detailed Description

Service discovery result

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_service_discovery_req_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_uuid_t	uuid Service UUID.
uint16_t	start_handle Discovery start handle.
uint16_t	end_handle Discovery end handle.
sf_ble_service_discovery_t	discovery_type Service discovery type.

Detailed Description

Service discovery request

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_discovery_req_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_uuid_t	uuid Characteristic UUID.
uint16_t	start_handle Discovery start handle.

uint16_t	end_handle Discovery end handle.
sf_ble_char_discovery_t	discovery_type Characteristic discovery type.

Detailed Description

Characteristic discovery request

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_discovery_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	char_handle Characteristic handle.
sf_ble_uuid_t	uuid Characteristic UUID.
uint16_t	value_handle Characteristic value handle.
sf_ble_char_property_t	property Characteristic property.

Detailed Description

Characteristic discovery result

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_desc_discovery_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	desc_handle Characteristic descriptor handle.
sf_ble_uuid_t	uuid Characteristic descriptor UUID.

Detailed Description

Characteristic descriptor discovery result

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_multiple_read_req_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	handles [SF_BLE_MAX_MULTI_CHAR_READ_CNT] Characteristic handles for multiple read.

uint8_t	expected_result_size [SF_BLE_MAX_MULTI_CHAR_READ_CNT] Expected result length in bytes.

Detailed Description

Multiple Characteristic descriptor read request

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_read_req_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_char_read_t	char_read_type Characteristic read type.
uint16_t	handle Characteristic value or descriptor handle.
sf_ble_uuid_t	uuid Characteristic UUID.
uint16_t	offset Offset for long Characteristic value.
sf_ble_char_multiple_read_req_t *	p_mul_read_req Pointer to multiple read Characteristic request.
uint16_t	handles_cnt

	Characteristic handles count for multiple read.

Detailed Description

Read Characteristic request

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_multiple_read_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t *	p_data [SF_BLE_MAX_MULTI_CHAR_READ_CNT] Pointer to Characteristic value.
uint16_t	data_len [SF_BLE_MAX_MULTI_CHAR_READ_CNT] Characteristic value length.

Detailed Description

Read multiple Characteristic response

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_read_by_uuid_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	handle [SF_BLE_MAX_CHAR_UUID_READ_CNT] Characteristic handles.
uint8_t *	p_data [SF_BLE_MAX_CHAR_UUID_READ_CNT] Pointer to Characteristic value.
uint16_t	data_len [SF_BLE_MAX_CHAR_UUID_READ_CNT] Characteristic value length.

Detailed Description

Read Characteristic by UUID response

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_read_by_handle_rsp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t *	p_data Pointer to Characteristic value.
uint16_t	data_len Characteristic value length.

Detailed Description

Read Characteristic by handle response

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_char_read_rsp_t Union Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_char_read_by_handle_rsp_t *	p_char_read_by_handle_rsp Response for read Characteristic by handle.
sf_ble_char_read_by_uuid_rsp_t *	p_char_read_by_uuid_rsp Response for read Characteristic by UUID.
sf_ble_char_multiple_read_rsp_t *	p_char_multiple_read_rsp Response for read multiple Characteristics.

Detailed Description

Read Characteristic response

The documentation for this union was generated from the following file:

- sf_ble_api.h

sf_ble_char_write_req_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_char_write_t	char_write_type Characteristic write type.
uint16_t	handle Characteristic value or descriptor handle.
uint8_t *	p_data Pointer to data.
uint16_t	offset Offset for long Characteristic value.
uint16_t	data_length Data length.
sf_ble_execute_write_t	auto_execute Automatic execute write flag.

Detailed Description

Write Characteristic request

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_gatt_notif_ind_event_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	handle

	Characteristic value or descriptor handle.
uint8_t *	p_data Pointer to data.
uint16_t	data_length Data length.

Detailed Description

Notification/Indication event data

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_write_cmd_event_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint16_t	handle Characteristic value or descriptor handle.
uint16_t	offset Offset for long Characteristic value.
sf_ble_write_response_t	response_required Write response required or not.
uint8_t *	p_data Pointer to data.

uint16_t	data_length Data length.

Detailed Description

Write command event data

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

void *	p_driver_handle Storage for information needed for each BLE device driver in the system.

Detailed Description

BLE Framework control structure

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

uint8_t	bd_addr [SF_BLE_ADDR_LEN] BLE address.
sf_ble_addr_type_t	own_addr_type self address type
uint8_t	max_slaves Maximum slaves allowed to be connected.
uint8_t	update_bd_addr Set this to true to update bluetooth address during SF_BLE_Open.
uint16_t	scan_interval BLE scan interval for receiving advertisement.
uint16_t	scan_window Period of time during which advertising data is received at the scan interval.
uint16_t	disc_time Duration for which the device remain discoverable.
uint16_t	con_interval Interval for transmitting and receiving data periodically after connection establishment.
uint16_t	slave_latency Period of time during which data is transmitted and received at the connection interval.
uint16_t	sup_timeout Link loss time-out.
void const *	p_extend

	Instance specific configuration.

Detailed Description

BLE configuration information

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

ssp_err_t(*	open)(sf_ble_ctrl_t *const p_ctrl, const sf_ble_cfg_t *p_cfg) Initializes the interface for data transfers. More...
ssp_err_t(*	close)(sf_ble_ctrl_t *const p_ctrl) De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	infoGet)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_info_t *p_ble_info) Get BLE module information like chipset information and RSSI value. More...
ssp_err_t(*	provisionGet)(sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t *p_ble_provisioning) Reads the current BLE Provisioning information. More...
ssp_err_t(*	provisionSet)(sf_ble_ctrl_t *const p_ctrl, const sf_ble_provisioning_t *p_ble_provisioning) Provisions the BLE Driver and set bonding and security modes as provisioned. More...

ssp_err_t(*	scan)(sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t *p_scan, uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info) Scans for available BLE devices and return the list to the caller. More...
ssp_err_t(*	advertisementStart)(sf_ble_ctrl_t *const p_ctrl, sf_ble_adv_info_t *const p_advt_info) Make the device discoverable by broadcasting device information to remote devices. More...
ssp_err_t(*	advertisementStop)(sf_ble_ctrl_t *const p_ctrl) Stop the device from being discoverable. More...
ssp_err_t(*	whitelistAdd)(sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr) Add specified devices to whitelist. More...
ssp_err_t(*	whitelistDel)(sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr) Remove specified devices from whitelist. More...
ssp_err_t(*	bondingStart)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_start_t *p_bonding_start) Initiate bonding process with remote BLE device and exchange security keys if enabled. More...
ssp_err_t(*	bondingResponse)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_response_t *p_bonding_resp) Respond to the bonding request from the remote BLE device. Send bonding response on reception of bonding indication. More...
ssp_err_t(*	startEncryption)(sf_ble_ctrl_t *const p_ctrl, sf_ble_sm_enc_info_t const *p_enc_info) Start encryption over a connection. More...
ssp_err_t(*	connect)(sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle) Connect to a remote BLE device. More...

ssp_err_t(*	listen)(sf_ble_ctrl_t *const p_ctrl) Listen for connection request from remote device. More...
ssp_err_t(*	authorization)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle) Indicates that the specified remote device has been authorized by user. More...
ssp_err_t(*	disconnect)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle) Terminate connection with remote BLE device. More...
ssp_err_t(*	setTxPower)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info) Sets the transmit power for the procedure specified by the connection handle. More...
ssp_err_t(*	gattAddCustomProfiles)(sf_ble_ctrl_t *const p_ctrl, sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len, sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len) Add Custom Profile to GATT Database. More...
ssp_err_t(*	gattServiceDiscovery)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt) Perform service discovery used by GATT client.Perform service discovery on remote GATT server and get results. More...
ssp_err_t(*	gattCharDiscovery)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt) Perform characteristics discovery used by GATT client.Perform characteristics discovery on remote GATT server and get results. More...
ssp_err_t(*	gattCharDescDiscovery)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle, sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t *const p_rsp_cnt)

	Perform characteristics descriptor discovery used by GATT client. More...
ssp_err_t(*	gattCharRead)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp) Perform read characteristic used by GATT client.Read characteristic value from remote GATT server. More...
ssp_err_t(*	gattCharWrite)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const p_char_write_req) Perform write characteristic used by GATT client.Write characteristic value on remote GATT server. More...
ssp_err_t(*	gattCharExecuteWrite)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag) Perform execute write on all pending write operations, used by GATT client.Execute or cancel all prepared writes of characteristics on remote GATT server. More...
ssp_err_t(*	gattCharWriteLocal)(sf_ble_ctrl_t *const p_ctrl, uint16_t char_handle, uint16_t data_length, uint8_t *const p_data) Perform local characteristic write used by GATT server.Writes local characteristic value on GATT server. More...
ssp_err_t(*	gattSendNotify)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send notification to GATT client, used by GATT server. More...
ssp_err_t(*	gattSendIndicate)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send indication to GATT client, used by GATT server.Send indication to remote GATT client. More...
ssp_err_t(*	gattWriteResponse)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t error_code) Send response to write operation received by GATT client, used by GATT server.Send response for write request received from remote GATT client. More...

ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Framework API structure. Implementations will use the following API.

Field Documentation

◆ advertisementStart

ssp_err_t(* sf_ble_api_t::advertisementStart) (sf_ble_ctrl_t *const p_ctrl, sf_ble_adv_info_t *const p_advt_info)

Make the device discoverable by broadcasting device information to remote devices.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_advt_info	Pointer to advertisement information structure

◆ advertisementStop

ssp_err_t(* sf_ble_api_t::advertisementStop) (sf_ble_ctrl_t *const p_ctrl)

Stop the device from being discoverable.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.

◆ authorization

ssp_err_t(* sf_ble_api_t::authorization) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle)

Indicates that the specified remote device has been authorized by user.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle.

◆ bondingResponse

ssp_err_t(* sf_ble_api_t::bondingResponse) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_response_t *p_bonding_resp)

Respond to the bonding request from the remote BLE device. Send bonding response on reception of bonding indication.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle
[in]	p_bd_addr	Pointer to BLE address
[in]	p_bonding_resp	Pointer to Bonding address

◆ bondingStart

ssp_err_t(* sf_ble_api_t::bondingStart) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_start_t *p_bonding_start)

Initiate bonding process with remote BLE device and exchange security keys if enabled.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle.
[in]	p_bd_addr	Pointer to BLE address

◆ close

ssp_err_t(* sf_ble_api_t::close) (sf_ble_ctrl_t *const p_ctrl)

De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.

◆ connect

ssp_err_t(* sf_ble_api_t::connect) (sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle)

Connect to a remote BLE device.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_conn	Pointer to connection information
[out]	p_handle	Pointer to connection handle

◆ disconnect

ssp_err_t(* sf_ble_api_t::disconnect) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle)

Terminate connection with remote BLE device.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle

◆ gattAddCustomProfiles

ssp_err_t(* sf_ble_api_t::gattAddCustomProfiles) (sf_ble_ctrl_t *const p_ctrl, sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len, sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len)

Add Custom Profile to GATT Database.

This function is called with list of service and characteristics which is to be added to GATT database. Services and characteristics which were previously added will be removed and newly passed services and characteristics will be added.

If services and characteristics were previously added and now those services and characteristics are to be removed and no new services and characteristics are to be added , then call this API with service and characteristics length as zero

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_svc_attr	List of Services to add
[in]	svc_attr_len	No of elements in services list
[in]	p_char_attr	List of Characteristics to add
[in]	char_attr_len	No of elements in Characteristics list

◆ gattCharDescDiscovery

ssp_err_t(* sf_ble_api_t::gattCharDescDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle, sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t *const p_rsp_cnt)

Perform characteristics descriptor discovery used by GATT client.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	start_handle	Start handle from set of handle ranges to be used in discovery
[in]	end_handle	End handle from set of handle ranges to be used in discovery
[out]	p_sf_ble_chardesc_dscv_rsp	Pointer to characteristics descriptor discovery response
[in,out]	p_rsp_cnt	Input Size specifying maximum number of characteristics descriptor discovery results which can be stored in response, output specifying number of characteristics descriptor discovery results stored in response

◆ gattCharDiscovery

ssp_err_t(* sf_ble_api_t::gattCharDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt)

Perform characteristics discovery used by GATT client.Perform characteristics discovery on remote GATT server and get results.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	p_sf_ble_char_dscv_req	Pointer to characteristics discovery request
[out]	p_sf_ble_char_dscv_rsp	Pointer to characteristics discovery response
[in,out]	p_rsp_cnt	Input Size specifying maximum number of characteristics discovery results which can be stored in response, output specifying number of characteristics discovery results stored in response

◆ gattCharExecuteWrite

ssp_err_t(* sf_ble_api_t::gattCharExecuteWrite) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag)

Perform execute write on all pending write operations, used by GATT client.Execute or cancel all prepared writes of characteristics on remote GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	execute_flag	Flag specifying whether to execute or cancel pending writes

◆ gattCharRead

ssp_err_t(* sf_ble_api_t::gattCharRead) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp)

Perform read characteristic used by GATT client.Read characteristic value from remote GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	p_char_read_req	Pointer to characteristic read request
[out]	p_char_read_rsp	Pointer to characteristic read response

◆ gattCharWrite

ssp_err_t(* sf_ble_api_t::gattCharWrite) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const p_char_write_req)

Perform write characteristic used by GATT client.Write characteristic value on remote GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	p_char_write_req	Pointer to characteristic write request

◆ gattCharWriteLocal

ssp_err_t(* sf_ble_api_t::gattCharWriteLocal) (sf_ble_ctrl_t *const p_ctrl, uint16_t char_handle, uint16_t data_length, uint8_t *const p_data)

Perform local characteristic write used by GATT server.Writes local characteristic value on GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	char_handle	Characteristic handle
[in]	data_length	Length of data to write
[in]	p_data	Pointer to data

◆ gattSendIndicate

ssp_err_t(* sf_ble_api_t::gattSendIndicate) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle)

Send indication to GATT client, used by GATT server.Send indication to remote GATT client.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	char_handle	Characteristic handle whose value will be indicated

◆ gattSendNotify

ssp_err_t(* sf_ble_api_t::gattSendNotify) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle)

Send notification to GATT client, used by GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	char_handle	Characteristic handle whose value will be notified

◆ gattServiceDiscovery

ssp_err_t(* sf_ble_api_t::gattServiceDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt)

Perform service discovery used by GATT client.Perform service discovery on remote GATT server and get results.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	p_sf_ble_svc_dscv_req	Pointer to service discovery request
[out]	p_sf_ble_svc_dscv_rsp	Pointer to service discovery response
[in,out]	p_rsp_cnt	Input Size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

◆ gattWriteResponse

ssp_err_t(* sf_ble_api_t::gattWriteResponse) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t error_code)

Send response to write operation received by GATT client, used by GATT server.Send response for write request received from remote GATT client.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Connection handle
[in]	handle	Characteristic handle used for write operation
[in]	error_code	Characteristic write operation error code to be sent in response

◆ infoGet

ssp_err_t(* sf_ble_api_t::infoGet) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_info_t *p_ble_info)

Get BLE module information like chipset information and RSSI value.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle
[out]	p_ble_info	Pointer to module information

◆ listen

ssp_err_t(* sf_ble_api_t::listen) (sf_ble_ctrl_t *const p_ctrl)

Listen for connection request from remote device.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.

◆ open

ssp_err_t(* sf_ble_api_t::open) (sf_ble_ctrl_t *const p_ctrl, const sf_ble_cfg_t *p_cfg)

Initializes the interface for data transfers.

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to BLE configuration structure.

◆ provisionGet

ssp_err_t(* sf_ble_api_t::provisionGet) (sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t *p_ble_provisioning)

Reads the current BLE Provisioning information.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[out]	p_ble_provisioning	current provisioning information

◆ provisionSet

ssp_err_t(* sf_ble_api_t::provisionSet) (sf_ble_ctrl_t *const p_ctrl, const sf_ble_provisioning_t *p_ble_provisioning)

Provisions the BLE Driver and set bonding and security modes as provisioned.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_ble_provisioning	Pointer to BLE provisioning structure

◆ scan

ssp_err_t(* sf_ble_api_t::scan) (sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t *p_scan, uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info)

Scans for available BLE devices and return the list to the caller.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[out]	p_scan	Pointer to scan information structure
[in,out]	p_cnt	Pointer to number of BLE devices scanned
[in]	p_scan_info	Pointer to scan information structure

◆ setTxPower

ssp_err_t(* sf_ble_api_t::setTxPower) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info)

Sets the transmit power for the procedure specified by the connection handle.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_handle	Pointer to connection handle. If transmit power is to be set before advertisement or connecting to remote device then pass connection handle as SF_BLE_TX_POWER_CONNECTION_HANDLE value else pass connection handle of remote device
[in]	p_tx_power_info	Pointer to TX power information

◆ startEncryption

ssp_err_t(* sf_ble_api_t::startEncryption) (sf_ble_ctrl_t *const p_ctrl, sf_ble_sm_enc_info_t const *p_enc_info)

Start encryption over a connection.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_enc_info	information for starting encryption

◆ versionGet

ssp_err_t(* sf_ble_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	pointer to memory location to return version number

◆ whitelistAdd

ssp_err_t(* sf_ble_api_t::whitelistAdd) (sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr)

Add specified devices to whitelist.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_bd_addr	Pointer to BLE address

◆ whitelistDel

ssp_err_t(* sf_ble_api_t::whitelistDel) (sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr)

Remove specified devices from whitelist.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.
[in]	p_bd_addr	Pointer to BLE address

The documentation for this struct was generated from the following file:

- sf_ble_api.h

sf_ble_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Framework Interface

#include <sf_ble_api.h>

Data Fields

sf_ble_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_ble_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_ble_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

BLE instance

The documentation for this struct was generated from the following file:

- sf_ble_api.h

5.1.2.6 SF BLE On-Board Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE On-Board Profile Framework Interface. More...

Data Structures

struct	sf_ble_onboard_profile_cccd_changed_t
struct	sf_ble_attr_info_t
struct	sf_ble_long_attr_info_t
struct	sf_ble_onboard_profile_cfg_t
struct	sf_ble_onboard_profile_ctrl_t
struct	sf_ble_onboard_profile_api_t

struct	sf_ble_onboard_profile_instance_t

Macros

#define	SF_BLE_ONBOARD_PROFILE_API_VERSION_MAJOR (2U)
#define	SF_BLE_ONBOARD_PROFILE_API_VERSION_MINOR (0U)
#define	SF_BLE_ATTM_MAX_VALUE (0x18)

Typedefs

typedef void(*	sf_ble_profile_callback_t) (sf_ble_event_info_t *ev)

Enumerations

enum	sf_onbp_t {
	SF_ONBP_ANS_SERVER, SF_ONBP_ANS_CLIENT,
	SF_ONBP_BAS_SERVER, SF_ONBP_BPS_SERVER,
	SF_ONBP_BPS_CLIENT, SF_ONBP_CTS_SERVER,
	SF_ONBP_DIS_SERVER, SF_ONBP_FMP_SERVER,
	SF_ONBP_FMP_CLIENT, SF_ONBP_HTP_SERVER,
	SF_ONBP_HTP_CLIENT, SF_ONBP_HRS_SERVER,
	SF_ONBP_HRS_CLIENT, SF_ONBP_HID_SERVER,
	SF_ONBP_HID_BHOST_CLIENT, SF_ONBP_HID_RHOST_CLIENT,
	SF_ONBP_IMA_SERVER, SF_ONBP_LLS_SERVER,
	SF_ONBP_LLS_CLIENT, SF_ONBP_NDCS_SERVER,
	SF_ONBP_PAPS_SERVER, SF_ONBP_PAPS_CLIENT,
	SF_ONBP_PXP_SERVER, SF_ONBP_PXP_CLIENT,
	SF_ONBP_RTUS_SERVER, SF_ONBP_SCPS_SERVER,
	SF_ONBP_SCPS_CLIENT, SF_ONBP_TIP_SERVER,
	SF_ONBP_TIP_CLIENT, SF_ONBP_TXP_SERVER
	}

enum	sf_ble_prf_sec_t {
	SF_BLE_PRF_SEC_NONE = 0x01, SF_BLE_PRF_SEC_UNAUTH = 0x02,
	SF_BLE_PRF_SEC_AUTH = 0x04, SF_BLE_PRF_SEC_AUTZ = 0x08,
	SF_BLE_PRF_SEC_ENC = 0x10
	}

enum	sf_ble_onbp_char_t { SF_BLE_ONBP_CHAR_HRP_HRCP, SF_BLE_ONBP_CHAR_HRP_CCCD_NTF_HRMEAS, SF_BLE_ONBP_CHAR_HRP_BSL, SF_BLE_ONBP_CHAR_ANP_SUP_NEW_ALERT, SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_NEW_ALERT, SF_BLE_ONBP_CHAR_ANP_SUP_UNREAD_ALERT, SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_UNREAD_ALERT_STATUS, SF_BLE_ONBP_CHAR_ANP_ALERT_NOTIFICATION_CTRL_POINT, SF_BLE_ONBP_CHAR_DIS_MANUF, SF_BLE_ONBP_CHAR_DIS_MODEL, SF_BLE_ONBP_CHAR_DIS_SERNB, SF_BLE_ONBP_CHAR_DIS_HWREV, SF_BLE_ONBP_CHAR_DIS_FWREV, SF_BLE_ONBP_CHAR_DIS_SWREV, SF_BLE_ONBP_CHAR_DIS_SYSID, SF_BLE_ONBP_CHAR_DIS_IEEE, SF_BLE_ONBP_CHAR_DIS_PNPID,

	SF_BLE_ONBP_CHAR_IAS_ALERT_LEVEL, SF_BLE_ONBP_CHAR_SCPS_CCCD_NTF_SCAN_REFRESH, SF_BLE_ONBP_CHAR_SCPS_SCAN_INTERVAL_WINDOW, SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME, SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME_VALUE, SF_BLE_ONBP_CHAR_CTS_LOCAL_TIME_INFORMATION, SF_BLE_ONBP_CHAR_CTS_REFERENCE_TIME_INFORMATION, SF_BLE_ONBP_CHAR_NDCS_TIME_WITH_DST, SF_BLE_ONBP_CHAR_TIME_UPDATE_CONTROL_POINT, SF_BLE_ONBP_CHAR_TIME_UPDATE_STATE, SF_BLE_ONBP_CHAR_CCCD_NTF_CURRENT_TIME, SF_BLE_ONBP_CHAR_CCCD_NTF_BATTERY_LEVEL, SF_BLE_ONBP_CHAR_PXPM_SET_ALERT, SF_BLE_ONBP_CHAR_PXPM_GET_ALERT_LVL, SF_BLE_ONBP_CHAR_PXPM_GET_TX_POWER_LVL, SF_BLE_ONBP_CHAR_HTP_TEMP_MEAS_IND, SF_BLE_ONBP_CHAR_HTP_INTERM_TEMP_NTF, SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_IND, SF_BLE_ONBP_CHAR_HTP_TEMP_TYPE, SF_BLE_ONBP_CHAR_HTP_MEAS_INTV, SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_RANGE, SF_BLE_ONBP_CHAR_CCCD_NTF_REPORT_INPUT, SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPORT, SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPORT_VALUE, SF_BLE_ONBP_CHAR_CCCD_NTF_KB_OUTPUT_REPORT_VALUE, SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT, SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT_VALUE, SF_BLE_ONBP_CHAR_PROTOCOL_MODE, SF_BLE_ONBP_CHAR_CONTROL_POINT, SF_BLE_ONBP_CHAR_BLP_CCCD_NTF_INTERM_CUFPRS, SF_BLE_ONBP_CHAR_BLP_CCCD_IND_BLDPRS_MEAS, SF_BLE_ONBP_CHAR_BLP_RD_BLS_BF, SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS, SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS_VALUE, SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING, SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING_VALUE, SF_BLE_ONBP_CHAR_RINGER_CONTROL_POINT }
enum	sf_ble_cccd_val_t { SF_BLE_CCCD_VAL_STOP_NTFIND, SF_BLE_CCCD_VAL_START_NTF, SF_BLE_CCCD_VAL_START_IND }

Detailed Description

RTOS-integrated SF BLE On-Board Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE On-Board Profile Framework.

Macro Definition Documentation

◆ SF_BLE_ATTM_MAX_VALUE

#define SF_BLE_ATTM_MAX_VALUE (0x18)
GATT Attribute Length

◆ SF_BLE_ONBOARD_PROFILE_API_VERSION_MAJOR

#define SF_BLE_ONBOARD_PROFILE_API_VERSION_MAJOR (2U)
SSP BSP Include files Framework Include files Major Version of the API defined in this file

◆ SF_BLE_ONBOARD_PROFILE_API_VERSION_MINOR

#define SF_BLE_ONBOARD_PROFILE_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

Typedef Documentation

◆ sf_ble_profile_callback_t

typedef void(* sf_ble_profile_callback_t) (sf_ble_event_info_t *ev)
User callback type for profile

Enumeration Type Documentation

◆ sf_ble_cccd_val_t

enum sf_ble_cccd_val_t	
Value for BLE CCCD Configuration	
Enumerator	
SF_BLE_CCCD_VAL_STOP_NTFIND	Stop Notification Indication.
SF_BLE_CCCD_VAL_START_NTF	Start Notification.
SF_BLE_CCCD_VAL_START_IND	Start Indication.

◆ sf_ble_onbp_char_t

enum sf_ble_onbp_char_t	
On-Board Profile GATT Characteristics Code	
Enumerator	
SF_BLE_ONBP_CHAR_HRP_HRCP	Heart Rate Control Point Characteristics (Property: Write), Refer sf_ble_prf_hrp_api_hrcp_t. Heart Rate Profile
SF_BLE_ONBP_CHAR_HRP_CCCD_NTF_HRMEAS	Heart Rate Measurement CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_hrp_api_hrmeas_t.
SF_BLE_ONBP_CHAR_HRP_BSL	Heart Rate Body Sensor Location Characteristics (Property: Read)
SF_BLE_ONBP_CHAR_ANP_SUP_NEW_ALERT	Supported New Alert Category Characteristics (Property: Read) Alert Notification Profile
SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_NEW_ALERT	New Alert CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_anp_api_new_alert_ntf_t.
SF_BLE_ONBP_CHAR_ANP_SUP_UNREAD_ALERT	Supported Unread Alert Category Characteristics (Property: Read)
SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_UNREAD_ALERT_STATUS	Unread Alert Status CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_anp_api_unread_alert_ntf_t.
SF_BLE_ONBP_CHAR_ANP_ALERT_NOTIFICATION_CTRL_POINT	Alert Notification Control Point (Property: Write), Refer sf_ble_anp_ancp_t.
SF_BLE_ONBP_CHAR_DIS_MANUF	Device Information Service Manufacturer Name String (Property: Read) Device Information Service Profile
SF_BLE_ONBP_CHAR_DIS_MODEL	Device Information Service Model Number String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SERNB	Device Information Service Serial number String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_HWREV	Device Information Service HW Revision String

	(Property: Read)
SF_BLE_ONBP_CHAR_DIS_FWREV	Device Information Service Fw Revision String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SWREV	Device Information Service SW Revision String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SYSID	Device Information Service system ID (Property: Read)
SF_BLE_ONBP_CHAR_DIS_IEEE	Device Information Service IEEE Certification (Property: Read)
SF_BLE_ONBP_CHAR_DIS_PNPID	Device Information PNPID, Used in services like HOGP (Property: Read)
SF_BLE_ONBP_CHAR_IAS_ALERT_LEVEL	Alert Level (Property: Write), Refer sf_ble_prf_ias_alert_type_t. Immediate Alert Service Profile
SF_BLE_ONBP_CHAR_SCPS_CCCD_NTF_SCAN_REFRESH	Scan Refresh CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_scps_scan_refresh_t. Scan Parameters Service Profile
SF_BLE_ONBP_CHAR_SCPS_SCAN_INTERVAL_WINDOW	Scan Interval Window (Property: Write), Refer sf_ble_prf_scps_scan_intv_t.
SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME	Current Time CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_cts_curr_time_ntf_t. Current Time Service Profile
SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME_VALUE	Used to read data for current time through read characteristics.
SF_BLE_ONBP_CHAR_CTS_LOCAL_TIME_INFORMATION	Local Time Information (Property: Read)
SF_BLE_ONBP_CHAR_CTS_REFERENCE_TIME_INFORMATION	Reference Time Information (Property: Read)
SF_BLE_ONBP_CHAR_NDCS_TIME_WITH_DST	Time with DST (Property: Read) Next DST Change Service Profile
SF_BLE_ONBP_CHAR_TIME_UPDATE_CONTROL_POINT	Time Update Control Point (Property: Write), Refer sf_ble_prf_rtus_time_updt_state_t. Reference Time Update Service Profile

SF_BLE_ONBP_CHAR_TIME_UPDATE_STATE	Time Update State (Property: Read)
SF_BLE_ONBP_CHAR_CCCD_NTF_CURRENT_TIME	Current Time CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_bas_battery_lvl_ntf_t. Time Information Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_BATTERY_LEVEL	Battery Level (Property: NTF, Read, Write), Refer sf_ble_prf_bas_battery_lvl_t. Battery Service Profile
SF_BLE_ONBP_CHAR_PXPM_SET_ALERT	PXP Alert char. Alert Level Service Profile
SF_BLE_ONBP_CHAR_PXPM_GET_ALERT_LVL	PXP Get Alert Level.
SF_BLE_ONBP_CHAR_PXPM_GET_TX_POWER_LVL	PXP Get TX Power Level.
SF_BLE_ONBP_CHAR_HTP_TEMP_MEAS_IND	HTPC temp. measurement indication CCCD Characteristics (Property: IND, Read, Write), Refer. Health Thermometer Profile
SF_BLE_ONBP_CHAR_HTP_INTERM_TEMP_NTF	HTPC intermediate temp. notification CCCD Characteristics (Property: NTF, Read, Write), Refer.
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_IND	HTPC temp. measurement interval indication CCCD Characteristics (Property: IND, Read, Write), Refer.
SF_BLE_ONBP_CHAR_HTP_TEMP_TYPE	HTPC temp. temperature type(Property: Read)
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV	HTPC temp. measurement interval.
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_RANGE	HTPC temp. measurement interval valid range.
SF_BLE_ONBP_CHAR_CCCD_NTF_REPORT_INPUT	Report Input CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t. HID Over GATT Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPORT	Keyboard Input Report CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPO	

RT_VALUE	Used to read Keyboard Input record value through read characteristics.
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_OUTPUT_REPORT_VALUE	Used to read Keyboard Output record value through read characteristics.
SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT	Mouse Input Report CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT_VALUE	Used to read Mouse Input record value through read characteristics.
SF_BLE_ONBP_CHAR_PROTOCOL_MODE	HID boot host protocol mode (Property: Read, Write), Refer sf_ble_prf_hid_protocol_mode_t.
SF_BLE_ONBP_CHAR_CONTROL_POINT	HID report host write control point (Property: Read, Write), Refer sf_ble_prf_hid_ctrl_point_val_t.
SF_BLE_ONBP_CHAR_BLP_CCCD_NTF_INTERM_CUFPRS	Read Cuff Pressure measurement CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_blp_meas_info_t. Blood Pressure profile
SF_BLE_ONBP_CHAR_BLP_CCCD_IND_BLDPRS_MEAS	Blood Pressure Measurement CCCD Characteristics (Property: IND, Read, Write), Refer sf_ble_blp_meas_info_t.
SF_BLE_ONBP_CHAR_BLP_RD_BLS_BF	Read Blood Pressure Feature.
SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS	Alert Status (Property: NTF, Read, Write), Refer sf_ble_prf_alert_status. Phone Alert Status Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS_VALUE	Used to read Alert Status value through read characteristics.
SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING	Ringer Settings (Property: NTF, Read, Write), Refer sf_ble_prf_ringer_setting_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING_VALUE	Used to read Ringer Settings value through read characteristics.
SF_BLE_ONBP_CHAR_RINGER_CONTROL_POINT	PAPS Ringer control point write characteristics.

◆ sf_ble_prf_sec_t

enum sf_ble_prf_sec_t	
BLE Profile Security type	
Enumerator	
SF_BLE_PRF_SEC_NONE	No Security.
SF_BLE_PRF_SEC_UNAUTH	Unauthenticated Pairing.
SF_BLE_PRF_SEC_AUTH	Authenticated pairing.
SF_BLE_PRF_SEC_AUTZ	Requires Authorized.
SF_BLE_PRF_SEC_ENC	Encrypted communication.

◆ sf_onbp_t

enum sf_onbp_t	
BLE On-Board Profile types	
Enumerator	
SF_ONBP_ANS_SERVER	Alert Notification Service Server.
SF_ONBP_ANS_CLIENT	Alert Notification Service Client.
SF_ONBP_BAS_SERVER	Battery Service Server.
SF_ONBP_BPS_SERVER	Blood Pressure Service Server.
SF_ONBP_BPS_CLIENT	Blood Pressure Service Client.
SF_ONBP_CTS_SERVER	Current Time Service Server.
SF_ONBP_DIS_SERVER	Device Information Service Server.
SF_ONBP_FMP_SERVER	Find Me Service Server.
SF_ONBP_FMP_CLIENT	Find Me Service Client.
SF_ONBP_HTP_SERVER	Health Thermometer Service Server.
SF_ONBP_HTP_CLIENT	Health Thermometer Service Client.
SF_ONBP_HRS_SERVER	Heart Rate Service Server.

SF_ONBP_HRS_CLIENT	Heart Rate Service Client.
SF_ONBP_HID_SERVER	HID Over GATT Service Server.
SF_ONBP_HID_BHOST_CLIENT	HID Over GATT Boot Host Service Client.
SF_ONBP_HID_RHOST_CLIENT	HID Over GATT Report Host Service Client.
SF_ONBP_IMA_SERVER	Immediate Alert Service Server.
SF_ONBP_LLS_SERVER	Link Loss Service Server.
SF_ONBP_LLS_CLIENT	Link Loss Service Client.
SF_ONBP_NDCS_SERVER	Next Daylight Savings Change Service Server.
SF_ONBP_PAPS_SERVER	Phone Alert Service Server.
SF_ONBP_PAPS_CLIENT	Phone Alert Service Client.
SF_ONBP_PXP_SERVER	Proximity Service Server.
SF_ONBP_PXP_CLIENT	Proximity Service Client.
SF_ONBP_RTUS_SERVER	Reference Time Update Service Server.
SF_ONBP_SCPS_SERVER	Scan Parameter Service Server.
SF_ONBP_SCPS_CLIENT	Scan Parameter Service Client.
SF_ONBP_TIP_SERVER	Time Information Service Server.
SF_ONBP_TIP_CLIENT	Time Information Service Client.
SF_ONBP_TXP_SERVER	Transmit Power Service Server.

sf_ble_onboard_profile_cccd_changed_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

sf_ble_conn_handle_t	conn_handle Connection handle.
sf_ble_onbp_char_t	char_code CCCD type that has been changed by the remote node.
sf_ble_cccd_val_t	cccd_val CCCD value.
uint8_t	inst_idx Instance index, Applicable for HOGP.

Detailed Description

BLE Profile CCCD Changed indication data

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_attr_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

uint8_t	len data length
uint8_t	data [SF_BLE_ATTM_MAX_VALUE] data

Detailed Description

BLE Profile Read Char data

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_long_attr_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

uint8_t	val_len size of the value data
uint8_t	reserved Reserved.
uint16_t	attr_hdl Attribute handle.
uint8_t	value [SF_BLE_ATTM_MAX_VALUE] actual value pairs

Detailed Description

BLE Profile Read Char data (Long type)

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_onboard_profile_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

sf_ble_instance_t const *	p_low_lvl_ble Low level BLE Interface.
void *	p_extend Extended configuration.

Detailed Description

BLE On-Board Profile configuration information

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_onboard_profile_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

sf_ble_instance_t *	p_low_lvl_ble Low level BLE Framework information needed by On-Board Profile.

Detailed Description

BLE On-Board Profile Framework control structure

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_onboard_profile_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

ssp_err_t(*	open)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, const sf_ble_onboard_profile_cfg_t *p_cfg) Initializes the interface for data transfers. More...
ssp_err_t(*	close)(sf_ble_onboard_profile_ctrl_t *const p_ctrl) De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	onbpEnable)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_profile_callback_t p_prf_cb, sf_ble_prf_sec_t sec) Enable On-Board Profile Enables On-Board profile on given connection handle with specified security type. Registers user callback for the profile. More...
ssp_err_t(*	onbpDisable)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile) Disable On-Board Profile Disables On-Board profile on given connection handle and unregisters user callback. More...
ssp_err_t(*	onbpServerWriteData)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data) Update Server Local Database Update Local GATT database of Profile Server. More...
ssp_err_t(*	onbpServerSendNotification)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

	Send notification from Server Sends Notification which will be data specific to On-Board Profile to Client. More...
ssp_err_t(*	onbpServerSendIndication)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data) Send Indication from Server Sends Indication which contains profile specific data to client. More...
ssp_err_t(*	onbpClientWriteCCCD)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t cccd_char, sf_ble_cccd_val_t cccd_val) Writes CCCD configuration in Server This API writes CCCD configuration of Server and which enables or disables notification or indication from server. More...
ssp_err_t(*	onbpClientWriteChar)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data) Writes GATT characteristics with data passed Writes the GATT characteristics in the Server with the data passed. More...
ssp_err_t(*	onbpClientReadChar)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics) Reads GATT characteristics from Server Reads GATT characteristics from the Server for the profile specified. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

BLE Configuration, Control and API structures Framework API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_ble_onboard_profile_api_t::close) (sf_ble_onboard_profile_ctrl_t *const p_ctrl)

De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in]	p_ctrl	Pointer to the control block for the BLE module.

◆ onbpClientReadChar

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientReadChar) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics)

Reads GATT characteristics from Server Reads GATT characteristics from the Server for the profile specified.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type
[in]	characteristics	Profile characteristics

◆ onbpClientWriteCCCD

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteCCCD) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t cccd_char, sf_ble_cccd_val_t cccd_val)

Writes CCCD configuration in Server This API writes CCCD configuration of Server and which enables or disables notification or indication from server.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Parameter_Description
[in]	cccd_char	CCCD Code
[in]	cccd_val	Configuration data of CCCD

◆ onbpClientWriteChar

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteChar) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

Writes GATT characteristics with data passed Writes the GATT characteristics in the Server with the data passed.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type
[in]	characteristics	GATT characteristics code
[in]	p_data	Pointer to data

◆ onbpDisable

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpDisable) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile)

Disable On-Board Profile Disables On-Board profile on given connection handle and unregisters user callback.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type to disable

◆ onbpEnable

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpEnable) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_profile_callback_t p_prf_cb, sf_ble_prf_sec_t sec)

Enable On-Board Profile Enables On-Board profile on given connection handle with specified security type. Registers user callback for the profile.

Parameters

[in,out]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type to enable
[in]	p_prf_cb	User callback for Profile
[in]	sec	Security type for profile

◆ onbpServerSendIndication

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendIndication) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

Send Indication from Server Sends Indication which contains profile specific data to client.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type
[in]	characteristics	Profile characteristics
[in]	p_data	Pointer to data

◆ onbpServerSendNotification

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendNotification) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

Send notification from Server Sends Notification which will be data specific to On-Board Profile to Client.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type
[in]	characteristics	Profile characteristics
[in]	p_data	Pointer to data

◆ onbpServerWriteData

ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerWriteData) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

Update Server Local Database Update Local GATT database of Profile Server.

Parameters

[in]	p_ctrl	Pointer to control structure for BLE
[in]	p_handle	Pointer to connection handle
[in]	profile	Profile type
[in]	characteristics	Profile characteristics
[in]	p_data	Pointer to data

◆ open

ssp_err_t(* sf_ble_onboard_profile_api_t::open) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, const sf_ble_onboard_profile_cfg_t *p_cfg)
Initializes the interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer. **Parameters** [in,out] p_ctrl — Pointer to user-provided storage for the control block. [in] p_cfg — Pointer to BLE configuration structure.

◆ versionGet

ssp_err_t(* sf_ble_onboard_profile_api_t::versionGet) (ssp_version_t *const p_version)
Gets version and stores it in provided pointer p_version. **Parameters** [out] p_version — pointer to memory location to return version number

---

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

sf_ble_onboard_profile_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE On-Board Profile Framework Interface

#include <sf_ble_onboard_profile_api.h>

Data Fields

sf_ble_onboard_profile_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

sf_ble_onboard_profile_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_ble_onboard_profile_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

Instance structure

The documentation for this struct was generated from the following file:

- sf_ble_onboard_profile_api.h

5.1.2.7 SF BLE Alert Notification Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Alert Notification Profile Framework Interface. More...

Data Structures

struct	sf_ble_anp_ancp_t
struct	sf_ble_anp_ancp_change_t
struct	sf_ble_anp_api_new_alert_t
struct	sf_ble_anp_api_new_alert_ntf_t
struct	sf_ble_anp_api_unread_alert_t
struct	sf_ble_anp_api_unread_alert_ntf_t

Macros

#define	SF_BLE_ANP_ALT_TEXT_MAX (18U)

Enumerations

enum	sf_ble_prf_anpc_event_t { SF_BLE_PRF_ANPC_EVENT_NONE,

	SF_BLE_PRF_ANPC_EVENT_NEW_ALT_NTF, SF_BLE_PRF_ANPC_EVENT_UNREAD_ALT_NTF, SF_BLE_PRF_ANPC_EVENT_READ_CHAR_RES }
enum	sf_ble_prf_anps_event_t { SF_BLE_PRF_ANPS_EVENT_NONE, SF_BLE_PRF_ANPS_EVENT_ALT_NF_CP_IND, SF_BLE_PRF_ANPS_EVENT_CCCD_NTF_IND }

enum	sf_ble_prf_anp_category_id {
	SF_BLE_PRF_ANP_CATEGORY_ID_SIMPLE_ALERT,
	SF_BLE_PRF_ANP_CATEGORY_ID_EMAIL,
	SF_BLE_PRF_ANP_CATEGORY_ID_NEWS,
	SF_BLE_PRF_ANP_CATEGORY_ID_CALL,
	SF_BLE_PRF_ANP_CATEGORY_ID_MISSED_CALL,
	SF_BLE_PRF_ANP_CATEGORY_ID_SMS_MMS,
	SF_BLE_PRF_ANP_CATEGORY_ID_VOICE_MAIL,
	SF_BLE_PRF_ANP_CATEGORY_ID_SCHEDULE,
	SF_BLE_PRF_ANP_CATEGORY_ID_HIGH_PRIORITY_ALERT,
	SF_BLE_PRF_ANP_CATEGORY_ID_INSTANT_MESSAGE,
	SF_BLE_PRF_ANP_CATEGORY_ID_ALL
	}

enum	sf_ble_prf_anp_cmd_id_t {
	SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_ENABLE,
	SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_ENABLE,
	SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_DISABLE,
	SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_DISABLE,
	SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_NTF_REQ,
	SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_NTF_REQ
	}

Detailed Description

RTOS-integrated SF BLE Alert Notification Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Alert Notification Profile Framework.

Macro Definition Documentation

◆ SF_BLE_ANP_ALT_TEXT_MAX

#define SF_BLE_ANP_ALT_TEXT_MAX (18U)
Buffer size of data for Alert Notification

Enumeration Type Documentation

◆ sf_ble_prf_anp_category_id

enum sf_ble_prf_anp_category_id	
Alert Notification Control Point Category ID	
Enumerator	
SF_BLE_PRF_ANP_CATEGORY_ID_SIMPLE_ALERT	Simple Alert: General text alert or non-text alert.
SF_BLE_PRF_ANP_CATEGORY_ID_EMAIL	Email Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_NEWS	News feeds Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_CALL	Incoming Call Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_MISSED_CALL	Missed Call Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_SMS_MMS	SMS/MMS Message Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_VOICE_MAIL	Voice Mail Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_SCHEDULE	Alert occurred on calendar, planner.
SF_BLE_PRF_ANP_CATEGORY_ID_HIGH_PRIORITY_ALERT	High Prioritized Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_INSTANT_MESSAGE	Incoming Instant Messages.
SF_BLE_PRF_ANP_CATEGORY_ID_ALL	All Supported Categories.

◆ sf_ble_prf_anp_cmd_id_t

enum sf_ble_prf_anp_cmd_id_t	
Alert Notification Control Point Command ID	
Enumerator	
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_ENABLE	Enable New Incoming Alert Notification.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_ENABLE	Enable Unread Category Status Notification.
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_DISABLE	Disable New Incoming Alert Notification.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_DISABLE	Disable Unread Category Status Notification.
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_NTF_REQ	Notify New Incoming Alert immediately.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_NTF_REQ	Notify Unread Category Status immediately.

◆ sf_ble_prf_anpc_event_t

enum sf_ble_prf_anpc_event_t	
Profile Client user events	
Enumerator	
SF_BLE_PRF_ANPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_ANPC_EVENT_NEW_ALT_NTF	New Alert Data received event, Refer sf_ble_anp_api_new_alert_ntf_t.
SF_BLE_PRF_ANPC_EVENT_UNREAD_ALT_NTF	Unread Alert Data received event, Refer sf_ble_anp_api_unread_alert_ntf_t.
SF_BLE_PRF_ANPC_EVENT_READ_CHAR_RES	Read Char Complete Event.

◆ sf_ble_prf_anps_event_t

enum sf_ble_prf_anps_event_t	
Alert notification server profile user events	
Enumerator	
SF_BLE_PRF_ANPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_ANPS_EVENT_ALT_NF_CP_IND	Alert Notification Control Point Changed indication, Refer sf_ble_anp_ancp_change_t.
SF_BLE_PRF_ANPS_EVENT_CCCD_NTF_IND	CCCD Notification Setting change event.

sf_ble_anp_ancp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_prf_anp_cmd_id_t	command_id Command type.
sf_ble_prf_anp_category_id	category_id Category on which to act.

Detailed Description

Write Characteristics data for Alert Notification Control Point

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

sf_ble_anp_ancp_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_anp_ancp_t	control_point_value Control point value.

Detailed Description

Alert Notification Control Point Change Indication for Sensor

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

sf_ble_anp_api_new_alert_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_prf_anp_category_id	category_id Category ID.
uint8_t	alert_num Number of New Alert.
uint8_t	text_size Text Size.

uint8_t	text [SF_BLE_ANP_ALT_TEXT_MAX] Actual Text.

Detailed Description

Notification Data for New Alert, also used for sending notification from server

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

sf_ble_anp_api_new_alert_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_anp_api_new_alert_t	new_alert New Alert data.

Detailed Description

Notification Data received for New Alert

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

sf_ble_anp_api_unread_alert_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification

Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_prf_anp_category_id	category_id Category ID.
uint8_t	unread_count Number of Unread Alert.

Detailed Description

Notification Data for Unread Alert, also used for sending notification from server

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

sf_ble_anp_api_unread_alert_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Alert Notification Profile Framework Interface

#include <sf_ble_prf_anp_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_anp_api_unread_alert_t	alert Unread Alert data.

Detailed Description

Notification Data received for Unread Alert

The documentation for this struct was generated from the following file:

- sf_ble_prf_anp_api.h

5.1.2.8 SF BLE Battery Service Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Battery Service Profile Framework Interface. More...

Data Structures

struct	sf_ble_bas_battery_lvl_ntf_t

Variables

SSP_HEADER typedef uint8_t	sf_ble_prf_bas_battery_lvl_t

Detailed Description

RTOS-integrated SF BLE Battery Service Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Battery Service Profile Framework.

Variable Documentation

◆ **sf_ble_prf_bas_battery_lvl_t**

SSP_HEADER typedef uint8_t sf_ble_prf_bas_battery_lvl_t
Battery Level

sf_ble_bas_battery_lvl_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Battery Service Profile Framework Interface

#include <sf_ble_prf_bas_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_bas_battery_lvl_t	batt_lvl Battery Level.

Detailed Description

Control Point Change Indication for Sensor

The documentation for this struct was generated from the following file:

- sf_ble_prf_bas_api.h

5.1.2.9 SF BLE Blood Pressure Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Blood Pressure Profile Framework Interface. More...

Data Structures

struct	sf_ble_blp_meas_info_t
struct	sf_ble_blp_meas_recv_data_t

Enumerations

enum	sf_ble_prf_blpc_event_t { SF_BLE_PRF_BLPC_EVENT_NONE, SF_BLE_PRF_BLPC_EVENT_MEAS_NTF_IND, SF_BLE_PRF_BLPC_EVENT_WRITE_CHAR_RES, SF_BLE_PRF_BLPC_EVENT_READ_CHAR_RES }
enum	sf_ble_prf_blps_event_t { SF_BLE_PRF_BLPS_EVENT_NONE, SF_BLE_PRF_BLPS_EVENT_NTFIND_IND }

Detailed Description

RTOS-integrated SF BLE Blood Pressure Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Blood Pressure Profile Framework.

Enumeration Type Documentation

◆ sf_ble_prf_blpc_event_t

enum sf_ble_prf_blpc_event_t	
Profile user client events	
Enumerator	
SF_BLE_PRF_BLPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_BLPC_EVENT_MEAS_NTF_IND	BLE user event indicating BLPC profile measurement notification.
SF_BLE_PRF_BLPC_EVENT_WRITE_CHAR_RES	BLE user event indicating BLPC profile read char response.
SF_BLE_PRF_BLPC_EVENT_READ_CHAR_RES	BLE user event indicating BLPC write char response.

◆ sf_ble_prf_blps_event_t

enum sf_ble_prf_blps_event_t	
Profile user server events	
Enumerator	
SF_BLE_PRF_BLPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_BLPS_EVENT_NTFIND_IND	BLE user event indicating notification indication.

sf_ble_blp_meas_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Blood Pressure Profile Framework Interface

#include <sf_ble_prf_blp_api.h>

Data Fields

uint8_t	flag_stable_meas

	Stable or intermediary type of measurements. Set to 0 if intermediate measurement.
uint8_t	flags flags
int16_t	press_val1 blood pressure value - Systolic or cuff pressure. Cuff pressure has to be filled here
int16_t	press_val2 blood pressure value - Diastolic or subfield1
int16_t	press_val3 blood pressure value - MAP or subfield2
sf_ble_prf_cts_date_time_t	stamp time stamp
int16_t	rate pulse rate
uint8_t	id user ID
uint8_t	reserved Reserved.
uint16_t	meas_sts measurement status

Detailed Description

Blood Pressure Measurements Info

The documentation for this struct was generated from the following file:

- sf_ble_prf_blp_api.h

sf_ble_blp_meas_recv_data_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Blood Pressure Profile Framework Interface

#include <sf_ble_prf_blp_api.h>

Data Fields

uint16_t	conhdl Connection handle.
sf_ble_onbp_char_t	char_code Characteristic code.
sf_ble_blp_meas_info_t	meas_info BLP measurement data.

Detailed Description

Blood Pressure Measurements Data Received from Server

The documentation for this struct was generated from the following file:

- sf_ble_prf_blp_api.h

5.1.2.10 SF BLE Current Time Service Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Current Time Service Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_cts_date_time_t
struct	sf_ble_prf_cts_curr_time_t
struct	sf_ble_cts_local_time_t
struct	sf_ble_cts_ref_time_t
struct	sf_ble_cts_curr_time_ntf_t

Detailed Description

RTOS-integrated SF BLE Current Time Service Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Current Time Service Profile Framework.

sf_ble_prf_cts_date_time_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Current Time Service Profile Framework Interface

#include <sf_ble_prf_cts_api.h>

Data Fields

uint16_t	year Year value.
uint8_t	month Month value.
uint8_t	day Day value.
uint8_t	hour Hour value.

uint8_t	min Minute value.
uint8_t	sec Second value.
uint8_t	reserved Reserved.

Detailed Description

Current Time Service Date Time information

The documentation for this struct was generated from the following file:

- sf_ble_prf_cts_api.h

sf_ble_prf_cts_curr_time_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Current Time Service Profile Framework Interface

#include <sf_ble_prf_cts_api.h>

Data Fields

sf_ble_prf_cts_date_time_t	stamp Date time info.
uint8_t	day_of_week Day of week.
uint8_t	fractions256 Fraction value.
uint8_t	adjust_reason

	Adjust reason.
uint8_t	reserved Reserved.

Detailed Description

Current time information with Date Time

The documentation for this struct was generated from the following file:

- sf_ble_prf_cts_api.h

sf_ble_cts_local_time_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Current Time Service Profile Framework Interface

#include <sf_ble_prf_cts_api.h>

Data Fields

int8_t	time_zone Time Zone.
uint8_t	dst_offset DST Offset.

Detailed Description

Local Time Information structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_cts_api.h

sf_ble_cts_ref_time_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Current Time Service Profile Framework Interface

#include <sf_ble_prf_cts_api.h>

Data Fields

uint8_t	time_source Source of time.
uint8_t	accuracy Estimated accuracy of time compared to original time source.
uint8_t	days_since_update Days that passed since time was updated successfully from time source.
uint8_t	hours_since_update Time that passed since time was updated successfully from time source.

Detailed Description

Reference Time Information structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_cts_api.h

sf_ble_cts_curr_time_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Current Time Service Profile Framework Interface

#include <sf_ble_prf_cts_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_cts_curr_time_t	current_time heart rate measurement data

Detailed Description

Notification Data of Current Time received from server

The documentation for this struct was generated from the following file:

- sf_ble_prf_cts_api.h

5.1.2.11 SF BLE Find Me Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Find Me Profile Framework Interface. More...

Enumerations

enum	sf_ble_prf_fmpt_event_t { SF_BLE_PRF_FMPT_EVENT_NONE, SF_BLE_PRF_FMPT_EVENT_ALERT_LVL_CHANGED }

Detailed Description

RTOS-integrated SF BLE Find Me Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Find Me Profile Framework.

Enumeration Type Documentation

◆ sf_ble_prf_fmpt_event_t

enum sf_ble_prf_fmpt_event_t	
Profile Server user events	
Enumerator	
SF_BLE_PRF_FMPT_EVENT_NONE	BLE event unsupported.
SF_BLE_PRF_FMPT_EVENT_ALERT_LVL_CHANGED	BLE user event Alert level changed from locator, Refer sf_ble_ias_alert_lvl_change_t.

5.1.2.12 SF BLE HID Over GATT Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE HID Over GATT Profile Framework Interface. More...

Data Structures

union	sf_ble_prf_value_t
struct	sf_ble_prf_hid_report_desc_t
struct	sf_ble_prf_hid_report_ind_t
struct	sf_ble_prf_hid_change_event_t
struct	sf_ble_prf_dis_pnpid_t

Macros

#define	SF_BLE_PRF_HIDS_REPORT_MAX (32U)

Typedefs

typedef uint8_t	sf_ble_prf_hid_protocol_mode_t
typedef uint8_t	sf_ble_prf_hid_ctrl_point_val_t

Enumerations

enum	sf_ble_prf_hidd_event_t { SF_BLE_PRF_HIDD_EVENT_NONE, SF_BLE_PRF_HIDD_EVENT_REPORT_IND, SF_BLE_PRF_HIDD_EVENT_CFG_IND, SF_BLE_PRF_HIDD_EVENT_PROTO_MODE_CHG_EVT, SF_BLE_PRF_HIDD_EVENT_REPORT_EVT,

	SF_BLE_PRF_HIDD_EVENT_CP_CHANGED_EVT }

enum	sf_ble_prf_hid_event_t {
	SF_BLE_PRF_HID_EVENT_NONE,
	SF_BLE_HID_BHOST_EVENT_REPORT_NTF,
	SF_BLE_HID_BHOST_EVENT_READ_CHAR_RESP,
	SF_BLE_HID_RHOST_EVENT_REPORT_NTF,
	SF_BLE_HID_RHOST_EVENT_BATTERY_LVL_NTF,
	SF_BLE_HID_RHOST_EVENT_READ_CHAR_RESP,
	SF_BLE_HID_RHOST_EVENT_READ_LONG_CHAR_RESP
	}

enum	sf_ble_hidd_device_type_t { SF_BLE_HIDD_HID_DEVICE = 0x01, SF_BLE_HIDD_BOOT_KEYBOARD, SF_BLE_HIDD_BOOT_MOUSE }

Detailed Description

RTOS-integrated SF BLE HID Over GATT Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE HID Over GATT Profile Framework.

Macro Definition Documentation

◆ SF_BLE_PRF_HIDS_REPORT_MAX

#define SF_BLE_PRF_HIDS_REPORT_MAX (32U)
Maximum Number of reports in HID

Typedef Documentation

◆ sf_ble_prf_hid_ctrl_point_val_t

typedef uint8_t sf_ble_prf_hid_ctrl_point_val_t
HID Control Point Characteristics

◆ sf_ble_prf_hid_protocol_mode_t

typedef uint8_t sf_ble_prf_hid_protocol_mode_t
Protocol Mode Characteristics

Enumeration Type Documentation

◆ sf_ble_hidd_device_type_t

enum sf_ble_hidd_device_type_t	
HID Device types	
Enumerator	
SF_BLE_HIDD_HID_DEVICE	HID Device type.
SF_BLE_HIDD_BOOT_KEYBOARD	Boot Keyboard type.
SF_BLE_HIDD_BOOT_MOUSE	Boot Mouse type.

◆ sf_ble_prf_hid_event_t

enum sf_ble_prf_hid_event_t	
Profile Client user events	
Enumerator	
SF_BLE_PRF_HID_EVENT_NONE	Event not supported.
SF_BLE_HID_BHOST_EVENT_REPORT_NTF	Report value received from HID device, Refer sf_ble_prf_hid_report_ind_t. HID BHOST
SF_BLE_HID_BHOST_EVENT_READ_CHAR_RESP	read char response received from HID device
SF_BLE_HID_RHOST_EVENT_REPORT_NTF	Report value received from HID device, Refer sf_ble_prf_hid_report_ind_t. HID RHOST
SF_BLE_HID_RHOST_EVENT_BATTERY_LVL_NTF	Battery level received from HID device, Refer sf_ble_bas_battery_lvl_ntf_t.
SF_BLE_HID_RHOST_EVENT_READ_CHAR_RESP	Read char response received from HID device.
SF_BLE_HID_RHOST_EVENT_READ_LONG_CHAR_RESP	Long char read response received from HID device.

◆ sf_ble_prf_hidd_event_t

enum sf_ble_prf_hidd_event_t	
Profile Server user events	
Enumerator	
SF_BLE_PRF_HIDD_EVENT_NONE	Event not supported.
SF_BLE_PRF_HIDD_EVENT_REPORT_IND	Report value updated from Boot Host or Report Host, Refer sf_ble_prf_hid_report_ind_t.
SF_BLE_PRF_HIDD_EVENT_CFG_IND	CCCD change indication from Boot Host or Report Host.
SF_BLE_PRF_HIDD_EVENT_PROTO_MODE_CHG_EVT	protocol mode change event from Boot Host or Report Host, Refer sf_ble_prf_hid_change_event_t
SF_BLE_PRF_HIDD_EVENT_REPORT_EVT	report value update event from Boot Host or Report Host, Refer sf_ble_prf_hid_report_ind_t
SF_BLE_PRF_HIDD_EVENT_CP_CHANGED_EVT	suspend event from Report Host, Refer sf_ble_prf_hid_change_event_t

sf_ble_prf_value_t Union Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE HID Over GATT Profile Framework Interface

#include <sf_ble_prf_hid_api.h>

Data Fields

sf_ble_prf_hid_protocol_mode_t	protocol_mode_val Protocol Mode.
sf_ble_prf_hid_ctrl_point_val_t	control_point_val HID Control Point.

Detailed Description

HID Profile value changed by the client

The documentation for this union was generated from the following file:

- sf_ble_prf_hid_api.h

sf_ble_prf_hid_report_desc_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE HID Over GATT Profile Framework Interface

#include <sf_ble_prf_hid_api.h>

Data Fields

sf_ble_hidd_device_type_t	device_type Device type.
uint8_t	report_type Report type.
uint8_t	value [SF_BLE_PRF_HIDS_REPORT_MAX] Report values.
uint16_t	value_size Report size.

Detailed Description

HID Report structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_hid_api.h

sf_ble_prf_hid_report_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE HID Over GATT Profile Framework Interface

#include <sf_ble_prf_hid_api.h>

Data Fields

uint16_t	conhdl Connection handle.
uint8_t	inst_idx Instance Index.
uint8_t	reserved Reserved.
sf_ble_prf_hid_report_desc_t	report Report received from either BHOST or RHOST.

Detailed Description

HID Report Indication structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_hid_api.h

sf_ble_prf_hid_change_event_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE HID Over GATT Profile Framework Interface

#include <sf_ble_prf_hid_api.h>

Data Fields

uint16_t	conhdl

	Connection handle.
uint8_t	inst_idx Instance Index.
sf_ble_prf_value_t	ble_prf_value HID Profile value changed by the client.

Detailed Description

HID Profile Values Change structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_hid_api.h

sf_ble_prf_dis_pnpid_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE HID Over GATT Profile Framework Interface

#include <sf_ble_prf_hid_api.h>

Data Fields

uint8_t	vendorIdSource Vendor ID source.
uint16_t	vendorId Vendor ID.
uint16_t	productId Product ID.
uint16_t	productVersion Version of Product.

Detailed Description

Structure to set the current Device Information PnP Id characteristic value

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_hid_api.h

5.1.2.13 SF BLE Heart Rate Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Heart Rate Profile Framework Interface. More...

Data Structures

struct	sf_ble_hrp_api_hrmeas_t
struct	sf_ble_hrp_api_meas_ntf_t
struct	sf_ble_hrp_cp_change_t

Macros

#define	SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN (0x9U)

Typedefs

typedef uint16_t	sf_ble_prf_hrp_api_hrcp_t

Enumerations

enum	sf_ble_prf_hrpc_event_t { SF_BLE_PRF_HRPC_EVENT_NONE, SF_BLE_PRF_HRPC_EVENT_MEAS_NTF, SF_BLE_PRF_HRPC_EVENT_READ_CHAR_RES }
enum	sf_ble_prf_hrps_event_t { SF_BLE_PRF_HRPS_EVENT_NONE, SF_BLE_PRF_HRPS_EVENT_NTF_CHG_IND, SF_BLE_PRF_HRPS_EVENT_HRCP_CHG_IND }

Detailed Description

RTOS-integrated SF BLE Heart Rate Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Heart Rate Profile Framework.

Macro Definition Documentation

◆ SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN

#define SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN (0x9U)
Heart Rate RR Interval Buffer Length

Typedef Documentation

◆ sf_ble_prf_hrp_api_hrcp_t

typedef uint16_t sf_ble_prf_hrp_api_hrcp_t
Write Characteristics data for Heart Rate Control Point

Enumeration Type Documentation

◆ sf_ble_prf_hrpc_event_t

enum sf_ble_prf_hrpc_event_t	
Profile Client user events	
Enumerator	
SF_BLE_PRF_HRPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_HRPC_EVENT_MEAS_NTF	Heart Rate Measurement data received event, Refer sf_ble_hrp_api_meas_ntf_t.
SF_BLE_PRF_HRPC_EVENT_READ_CHAR_RES	Read Char Complete Event.

◆ sf_ble_prf_hrps_event_t

enum sf_ble_prf_hrps_event_t	
Profile Server user events	
Enumerator	
SF_BLE_PRF_HRPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_HRPS_EVENT_NTF_CHG_IND	CCCD Notification Setting Change Event.
SF_BLE_PRF_HRPS_EVENT_HRCP_CHG_IND	Heart Rate Control Point Changed Event, Refer sf_ble_hrp_cp_change_t.

sf_ble_hrp_api_hrmeas_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Heart Rate Profile Framework Interface

#include <sf_ble_prf_hrp_api.h>

Data Fields

uint8_t	flags HRP bit Flags.
uint8_t	rr_interval_num number of RR Interval
uint16_t	heart_rate_measure Heart Rate measurement value.
uint16_t	energy_expended Energy Expended in Kilo Joules from last time it was reset.
uint16_t	rr_interval [SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN] RR interval(s)

Detailed Description

Notification Data for Heart Rate Measurement, also used for sending notification from server

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_hrp_api.h

sf_ble_hrp_api_meas_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Heart Rate Profile Framework Interface

#include <sf_ble_prf_hrp_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_hrp_api_hrmeas_t	measurements_info heart rate measurement data

Detailed Description

Notification Data of Heart Rate measurement received from sensor

---

The documentation for this struct was generated from the following file:

- sf_ble_prf_hrp_api.h

sf_ble_hrp_cp_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Heart Rate Profile Framework Interface

#include <sf_ble_prf_hrp_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_hrp_api_hrcp_t	control_point_value Control point value.

Detailed Description

Control Point Change Indication for Sensor

The documentation for this struct was generated from the following file:

- sf_ble_prf_hrp_api.h

5.1.2.14 SF BLE Health Thermometer Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Health Thermometer Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_htp_temp_info_t
struct	sf_ble_prf_htp_temp_info_ind_t
struct	st_sf_ble_prf_htp_meas_intv_val_t

Enumerations

enum	sf_ble_event_prf_htpt_t { SF_BLE_EVENT_PRF_HTPT_NONE, SF_BLE_EVENT_PRF_HTPT_MEAS_INTV_CHG_IND, SF_BLE_EVENT_PRF_HTPT_CFG_IND }
enum	sf_ble_event_prf_htpc_t { SF_BLE_EVENT_PRF_HTPC_NONE, SF_BLE_EVENT_PRF_HTPC_TEMP_IND, SF_BLE_EVENT_PRF_HTPC_MEAS_INTV_IND, SF_BLE_PRF_HTPC_EVENT_READ_CHAR_RES }
enum	sf_ble_prf_htp_meas_state_t { SF_BLE_PRF_HTP_MEAS_STATE_IN_PROGRESS = 0, SF_BLE_PRF_HTP_MEAS_STATE_COMPLETE }
enum	sf_ble_prf_htp_temp_type_t { SF_BLE_PRF_HTP_TYPE_ARMPIT = 0, SF_BLE_PRF_HTP_TYPE_BODY, SF_BLE_PRF_HTP_TYPE_EAR, SF_BLE_PRF_HTP_TYPE_FINGER, SF_BLE_PRF_HTP_TYPE_GASTROINTESTINAL, SF_BLE_PRF_HTP_TYPE_MOUTH, SF_BLE_PRF_HTP_TYPE_RECTUM, SF_BLE_PRF_HTP_TYPE_TOE, SF_BLE_PRF_HTP_TYPE_TYMPANUM }

Detailed Description

RTOS-integrated SF BLE Health Thermometer Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Health Thermometer Profile

Framework.

Enumeration Type Documentation

◆ sf_ble_event_prf_htpc_t

enum sf_ble_event_prf_htpc_t	
HTP Collector (client) user events	
Enumerator	
SF_BLE_EVENT_PRF_HTPC_NONE	Event not supported.
SF_BLE_EVENT_PRF_HTPC_TEMP_IND	BLE user event indicating HTPC profile temperature value indication.
SF_BLE_EVENT_PRF_HTPC_MEAS_INTV_IND	BLE user event indicating HTPC profile measurement interval value indication.
SF_BLE_PRF_HTPC_EVENT_READ_CHAR_RES	BLE user event for read char response.

◆ sf_ble_event_prf_htpt_t

enum sf_ble_event_prf_htpt_t	
HTP thermometer (server) user events	
Enumerator	
SF_BLE_EVENT_PRF_HTPT_NONE	Event not supported.
SF_BLE_EVENT_PRF_HTPT_MEAS_INTV_CHG_IND	Thermometer measurement interval characteristic change indication.
SF_BLE_EVENT_PRF_HTPT_CFG_IND	BLE user event indicating HTPT profile CCCD change indication.

◆ sf_ble_prf_htp_meas_state_t

enum sf_ble_prf_htp_meas_state_t	
HTP Collector (client) user events	
Enumerator	
SF_BLE_PRF_HTP_MEAS_STATE_IN_PROGRESS	Measurement in progress.
SF_BLE_PRF_HTP_MEAS_STATE_COMPLETE	Measurement is complete.

◆ sf_ble_prf_htp_temp_type_t

enum sf_ble_prf_htp_temp_type_t	
HTP Temperature Types	
Enumerator	
SF_BLE_PRF_HTP_TYPE_ARMPIT	Temperature type Armpit.
SF_BLE_PRF_HTP_TYPE_BODY	Temperature type Body.
SF_BLE_PRF_HTP_TYPE_EAR	Temperature type Ear.
SF_BLE_PRF_HTP_TYPE_FINGER	Temperature type Finger.
SF_BLE_PRF_HTP_TYPE_GASTROINTESTINAL	Temperature type Gastrointestinal.
SF_BLE_PRF_HTP_TYPE_MOUTH	Temperature type Mouth.
SF_BLE_PRF_HTP_TYPE_RECTUM	Temperature type Rectum.
SF_BLE_PRF_HTP_TYPE_TOE	Temperature type Toe.
SF_BLE_PRF_HTP_TYPE_TYMPANUM	Temperature type Tympanum.

sf_ble_prf_htp_temp_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Health Thermometer Profile Framework Interface

#include <sf_ble_prf_htp_api.h>

Data Fields

uint8_t	flag_stable_meas Stable or intermediary type of temperature.
uint8_t	flags flags
int32_t	temp_val temp value

sf_ble_prf_cts_date_time_t	stamp time stamp
sf_ble_prf_htp_temp_type_t	type type
uint8_t	reserved Reserved.

Detailed Description

Health Thermometer Info

The documentation for this struct was generated from the following file:

- sf_ble_prf_htp_api.h

sf_ble_prf_htp_temp_info_ind_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Health Thermometer Profile Framework Interface

#include <sf_ble_prf_htp_api.h>

Data Fields

uint16_t	conhdl Connection handle.
sf_ble_prf_htp_temp_info_t	temp_info Thermometer info.

Detailed Description

Thermometer Information Indication

The documentation for this struct was generated from the following file:

- sf_ble_prf_htp_api.h

st_sf_ble_prf_htp_meas_intv_val_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Health Thermometer Profile Framework Interface

#include <sf_ble_prf_htp_api.h>

Data Fields

uint16_t	conhdl Connection handle.
uint16_t	intv Interval value.

Detailed Description

Measurement Interval value

The documentation for this struct was generated from the following file:

- sf_ble_prf_htp_api.h

5.1.2.15 SF BLE Immediate Alert Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Immediate Alert Profile Framework Interface. More...

Data Structures

struct	sb_ble_prf_ias_set_alert_t
struct	sf_ble_prf_ias_alert_lvl_change_t

Enumerations

enum	sf_ble_prf_ias_alert_type_t { SF_BLE_PRF_ALERT_TYPE_NONE, SF_BLE_PRF_ALERT_TYPE_MILD, SF_BLE_PRF_ALERT_TYPE_HIGH }
enum	sf_ble_prf_ias_svc_code_t { SF_BLE_SVC_SET_LK_LOSS_ALERT = 0x00, SF_BLE_SVC_SET_IMMDT_ALERT }

Detailed Description

RTOS-integrated SF BLE Immediate Alert Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Immediate Alert Profile Framework.

Enumeration Type Documentation

◆ sf_ble_prf_ias_alert_type_t

enum sf_ble_prf_ias_alert_type_t	
Alert Level Values	
Enumerator	
SF_BLE_PRF_ALERT_TYPE_NONE	No Alert.
SF_BLE_PRF_ALERT_TYPE_MILD	Mild Alert.
SF_BLE_PRF_ALERT_TYPE_HIGH	High Alert.

◆ sf_ble_prf_ias_svc_code_t

enum sf_ble_prf_ias_svc_code_t	
SVC codes	
Enumerator	
SF_BLE_SVC_SET_LK_LOSS_ALERT	Code for LLS Alert Level Char.
SF_BLE_SVC_SET_IMMDT_ALERT	Code for IAS Alert Level Char.

sb_ble_prf_ias_set_alert_t Struct Reference
Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Immediate Alert Profile Framework Interface

#include <sf_ble_prf_ias_api.h>

Data Fields

sf_ble_prf_ias_svc_code_t	svc_code SVC code.
sf_ble_prf_ias_alert_type_t	lvl Alert level.

Detailed Description

Alert level and type

The documentation for this struct was generated from the following file:

- sf_ble_prf_ias_api.h

sf_ble_prf_ias_alert_lvl_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Immediate Alert Profile Framework Interface

#include <sf_ble_prf_ias_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_ias_alert_type_t	alert_lvl Control point value.

Detailed Description

Alert Level Change Indication for Server

The documentation for this struct was generated from the following file:

- sf_ble_prf_ias_api.h

5.1.2.16 SF BLE Next DST Change Service Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Next DST Change Service Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_ndcs_time_dst_t

Detailed Description

RTOS-integrated SF BLE Next DST Change Service Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Next DST Change Service Profile Framework.

sf_ble_prf_ndcs_time_dst_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Next DST Change Service Profile Framework Interface

#include <sf_ble_prf_ndcs_api.h>

Data Fields

sf_ble_prf_cts_date_time_t	stamp Current time stamp.
uint8_t	dst_offset DST Offset.
uint8_t	reserved Reserved.

Detailed Description

Next Time with DST Information structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_ndcs_api.h

5.1.2.17 SF BLE Phone Alert Status Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Phone Alert Status Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_ringer_cp_change_t
struct	sf_ble_prf_ringer_setting_ntf_t
struct	sf_ble_prf_alert_status_ntf_t

Typedefs

typedef uint8_t	sf_ble_prf_alert_status

Enumerations

enum	sf_ble_prf_papsc_event_t { SF_BLE_PRF_PAPSC_EVENT_NONE, SF_BLE_PRF_PAPSC_EVENT_READ_CHAR_RES, SF_BLE_PRF_PAPSC_EVENT_RINGER_SETTING_IND, SF_BLE_PRF_PAPSC_EVENT_ALERT_STATUS_RES }
enum	sf_ble_prf_papss_event_t { SF_BLE_PRF_PAPSS_EVENT_NONE, SF_BLE_PRF_PAPSS_EVENT_RINGER_CP_IND, SF_BLE_PRF_PAPSS_EVENT_CFG_NTF_IND }
enum	sf_ble_prf_ringer_cp_t { SF_BLE_PRF_RINGER_CP_SILENT_MODE = 0x01, SF_BLE_PRF_RINGER_CP_MUTE_ONCE, SF_BLE_PRF_RINGER_CP_CANCEL_SILENT_MODE }
enum	sf_ble_prf_ringer_setting_t { SF_BLE_PRF_RINGER_SETTING_SILENT, SF_BLE_PRF_RINGER_SETTING_NORMAL }
enum	sf_ble_prf_alert_setting_t { SF_BLE_PRF_ALERT_NONE, SF_BLE_PRF_ALERT_RINGER, SF_BLE_PRF_ALERT_VIBRATOR, SF_BLE_PRF_ALERT_DISPLAY }

Detailed Description

RTOS-integrated SF BLE Phone Alert Status Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Phone Alert Status Profile Framework.

Typedef Documentation

◆ sf_ble_prf_alert_status

typedef uint8_t sf_ble_prf_alert_status
PASP Alert Status

Enumeration Type Documentation

◆ sf_ble_prf_alert_setting_t

enum sf_ble_prf_alert_setting_t	
PASP Alert status	
Enumerator	
SF_BLE_PRF_ALERT_NONE	No active alert.
SF_BLE_PRF_ALERT_RINGER	Ringer State is active.
SF_BLE_PRF_ALERT_VIBRATOR	Vibrate State is active.
SF_BLE_PRF_ALERT_DISPLAY	Display Alert Status State is active.

◆ sf_ble_prf_papsc_event_t

enum sf_ble_prf_papsc_event_t	
Profile Client user events	
Enumerator	
SF_BLE_PRF_PAPSC_EVENT_NONE	Event not supported.
SF_BLE_PRF_PAPSC_EVENT_READ_CHAR_RES	BLE user event indicating PAPS target alert indication received from locator.
SF_BLE_PRF_PAPSC_EVENT_RINGER_SETTING_IND	BLE user event indicating PAPS ringer setting indication, Refer sf_ble_prf_ringer_setting_ntf_t .
SF_BLE_PRF_PAPSC_EVENT_ALERT_STATUS_RES	BLE user event indicating PAPS alert status response, Refer sf_ble_prf_alert_status_ntf_t.

◆ sf_ble_prf_papss_event_t

enum sf_ble_prf_papss_event_t	
Profile Server user events	
Enumerator	
SF_BLE_PRF_PAPSS_EVENT_NONE	Event not supported.
SF_BLE_PRF_PAPSS_EVENT_RINGER_CP_IND	BLE user event indicating ringer control point indication, Refer sf_ble_prf_ringer_cp_change_t .
SF_BLE_PRF_PAPSS_EVENT_CFG_NTF_IND	BLE user event indicating notification indication.

◆ sf_ble_prf_ringer_cp_t

enum sf_ble_prf_ringer_cp_t	
Ringer Control Point value	
Enumerator	
SF_BLE_PRF_RINGER_CP_SILENT_MODE	Silent Mode.
SF_BLE_PRF_RINGER_CP_MUTE_ONCE	Mute Once.
SF_BLE_PRF_RINGER_CP_CANCEL_SILENT_MODE	Cancel Silent Mode.

◆ sf_ble_prf_ringer_setting_t

enum sf_ble_prf_ringer_setting_t	
Ringer Setting Value	
Enumerator	
SF_BLE_PRF_RINGER_SETTING_SILENT	Ringer Silent.
SF_BLE_PRF_RINGER_SETTING_NORMAL	Ringer Normal.

sf_ble_prf_ringer_cp_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Phone Alert Status Profile Framework Interface

#include <sf_ble_prf_paps_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_ringer_cp_t	ringer_cp Ringer control point value.

Detailed Description

Ringer Control Point value changed indication

The documentation for this struct was generated from the following file:

- sf_ble_prf_paps_api.h

sf_ble_prf_ringer_setting_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Phone Alert Status Profile Framework Interface

#include <sf_ble_prf_paps_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_ringer_setting_t	ringer_setting Ringer control point value.

Detailed Description

Ringer Setting notification received data

The documentation for this struct was generated from the following file:

- sf_ble_prf_paps_api.h

sf_ble_prf_alert_status_ntf_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Phone Alert Status Profile Framework Interface

#include <sf_ble_prf_paps_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl

	Connection handle.
sf_ble_prf_alert_status	alert_status Ringer control point value.

Detailed Description

Alert Status notification received data

The documentation for this struct was generated from the following file:

- sf_ble_prf_paps_api.h

5.1.2.18 SF BLE Proximity Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Proximity Profile Framework Interface. More...

Enumerations

enum	sf_ble_prf_pxpr_event_t { SF_BLE_PRF_PXPR_EVENT_NONE, SF_BLE_PRF_PXPR_EVENT_ALT_IND, SF_BLE_PRF_PXPR_EVENT_CMD_DISALLOWED_IND }
enum	sf_ble_prf_pxpm_event_t { SF_BLE_PRF_PXPM_EVENT_NONE, SF_BLE_PRF_PXPM_EVENT_READ_CHAR_RESP, SF_BLE_PRF_PXPM_EVENT_CMD_DISALLOWED_IND, SF_BLE_PRF_PXPM_EVENT_ERROR_IND }

Detailed Description

RTOS-integrated SF BLE Proximity Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Proximity Profile Framework.

Enumeration Type Documentation

◆ sf_ble_prf_pxpm_event_t

enum sf_ble_prf_pxpm_event_t	
PXP monitor (client) user events	
Enumerator	
SF_BLE_PRF_PXPM_EVENT_NONE	unsupported event
SF_BLE_PRF_PXPM_EVENT_READ_CHAR_RESP	BLE user event indicating PXPR profile alert indication.
SF_BLE_PRF_PXPM_EVENT_CMD_DISALLOWED_IND	command disallowed error received
SF_BLE_PRF_PXPM_EVENT_ERROR_IND	error received for command issued

◆ sf_ble_prf_pxpr_event_t

enum sf_ble_prf_pxpr_event_t	
PXP reporter (server) user events	
Enumerator	
SF_BLE_PRF_PXPR_EVENT_NONE	unsupported event
SF_BLE_PRF_PXPR_EVENT_ALT_IND	BLE user event indicating PXPR profile alert indication.
SF_BLE_PRF_PXPR_EVENT_CMD_DISALLOWED_IND	command disallowed error received

5.1.2.19 SF BLE Reference Time Update Service Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Reference Time Update Service Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_rtus_time_updt_state_t
struct	sf_ble_tip_cp_change_t

Variables

SSP_HEADER typedef	sf_ble_prf_tip_time_ctrl_point_t

uint8_t	

Detailed Description

RTOS-integrated SF BLE Reference Time Update Service Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Reference Time Update Service Profile Framework.

Variable Documentation

◆ sf_ble_prf_tip_time_ctrl_point_t

SSP_HEADER typedef uint8_t sf_ble_prf_tip_time_ctrl_point_t
Time Update Control point

sf_ble_prf_rtus_time_updt_state_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Reference Time Update Service Profile Framework Interface

#include <sf_ble_prf_rtus_api.h>

Data Fields

uint8_t	current_state Current state of Reference time.
uint8_t	update_result Result of update.

Detailed Description

Reference Time Update State structure

The documentation for this struct was generated from the following file:

- sf_ble_prf_rtus_api.h

sf_ble_tip_cp_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Reference Time Update Service Profile Framework Interface

#include <sf_ble_prf_rtus_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_tip_time_ctrl_point_t	control_point_value Time Update Control point value.

Detailed Description

Time Update Control Point Change Indication for Server

The documentation for this struct was generated from the following file:

- sf_ble_prf_rtus_api.h

5.1.2.20 SF BLE Scan Parameters Service Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Scan Parameters Service Profile Framework Interface. More...

Data Structures

struct	sf_ble_prf_scps_scan_intv_t
struct	sf_ble_scps_scan_intv_change_t

Typedefs

typedef uint8_t	sf_ble_prf_scps_scan_refresh_t

Enumerations

enum	sf_ble_prf_scps_event_t { SF_BLE_PRF_SCPS_EVENT_NONE, SF_BLE_PRF_SCPS_EVENT_INDNTF_IND, SF_BLE_PRF_SCPS_EVENT_CHG_EVT }

Detailed Description

RTOS-integrated SF BLE Scan Parameters Service Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Scan Parameters Service Profile Framework.

Typedef Documentation

◆ sf_ble_prf_scps_scan_refresh_t

typedef uint8_t sf_ble_prf_scps_scan_refresh_t
Scan Refresh Data value

Enumeration Type Documentation

◆ sf_ble_prf_scps_event_t

enum sf_ble_prf_scps_event_t	
Profile Server user events	
Enumerator	
SF_BLE_PRF_SCPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_SCPS_EVENT_INDNTF_IND	BLE Notification Setting Change Indication.
SF_BLE_PRF_SCPS_EVENT_CHG_EVT	BLE Scan Interval Changed Indication, Refer sf_ble_scps_scan_intv_change_t.

sf_ble_prf_scps_scan_intv_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Scan Parameters Service Profile Framework Interface

#include <sf_ble_prf_scps_api.h>

Data Fields

uint16_t	le_scan_interval scan interval value
uint16_t	le_scan_window scan window value

Detailed Description

Scan interval window characteristic data

The documentation for this struct was generated from the following file:

- sf_ble_prf_scps_api.h

sf_ble_scps_scan_intv_change_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Scan Parameters Service Profile Framework Interface

#include <sf_ble_prf_scps_api.h>

Data Fields

sf_ble_conn_handle_t	conhdl Connection handle.
sf_ble_prf_scps_scan_intv_t	scan_interval_window_val Scan Interval window.

Detailed Description

Scan interval window Change Indication for Sensor

The documentation for this struct was generated from the following file:

- sf_ble_prf_scps_api.h

5.1.2.21 SF BLE Time Information Profile Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF BLE Time Information Profile Framework Interface. More...

Data Structures

union	sf_ble_prf_tip_write_data_t

Enumerations

enum	sf_ble_prf_tipc_event_t { SF_BLE_PRF_TIPC_EVENT_NONE, SF_BLE_PRF_TIPC_EVENT_READ_CHAR_RES, SF_BLE_PRF_TIPC_EVENT_CURR_TIME_NTF }
enum	sf_ble_prf_tips_event_t { SF_BLE_PRF_TIPS_EVENT_NONE, SF_BLE_PRF_TIPS_EVENT_NTF_IND, SF_BLE_PRF_TIPS_EVENT_CP_IND }

Detailed Description

RTOS-integrated SF BLE Time Information Profile Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Time Information Profile Framework.

Enumeration Type Documentation

◆ **sf_ble_prf_tipc_event_t**

enum sf_ble_prf_tipc_event_t	
Time Profile Client events	
Enumerator	
SF_BLE_PRF_TIPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_TIPC_EVENT_READ_CHAR_RES	Read Char Complete Event.
SF_BLE_PRF_TIPC_EVENT_CURR_TIME_NTF	Current Time information received, Refer sf_ble_cts_curr_time_ntf_t.

◆ sf_ble_prf_tips_event_t

enum sf_ble_prf_tips_event_t	
Time Profile Server events	
Enumerator	
SF_BLE_PRF_TIPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_TIPS_EVENT_NTF_IND	CCCD Notification Received event.
SF_BLE_PRF_TIPS_EVENT_CP_IND	Time Update control point changed, Refer sf_ble_prf_tip_time_ctrl_point_t.

sf_ble_prf_tip_write_data_t Union Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF BLE Time Information Profile Framework Interface

#include <sf_ble_prf_tip_api.h>

Data Fields

sf_ble_prf_cts_curr_time_t	current_time Current Time Information.
sf_ble_cts_local_time_t	local_time Local Time Information.
sf_ble_cts_ref_time_t	ref_time Reference Time Information.
sf_ble_prf_ndcs_time_dst_t	next_dst Next DST Information.
sf_ble_prf_rtus_time_updt_state_t	update_state Update Status Information.

Detailed Description

Write Local Database Information

The documentation for this union was generated from the following file:

- sf_ble_prf_tip_api.h

5.1.2.22 Block Media Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated File system Interface to access Synergy block media devices. More...

Data Structures

struct	sf_block_media_cfg_t
struct	sf_block_media_api_t
struct	sf_block_media_instance_t

Typedefs

typedef void	sf_block_media_ctrl_t

Detailed Description

RTOS-integrated File system Interface to access Synergy block media devices.

The interface provides an adaption layer from the FileX I/O to block media devices.

Summary

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also FileX Interface description: FileX on Block Media

Typedef Documentation

◆ sf_block_media_ctrl_t

typedef void sf_block_media_ctrl_t

Block media framework control block. Allocate an instance specific control block to pass into the block media framework API calls.

Implemented as

- sf_block_media_sdmmc_instance_ctrl_t
- sf_block_media_ram_instance_ctrl_t
- sf_block_media_qspi_instance_ctrl_t
- sf_block_media_levelx_nor_instance_ctrl_t

sf_block_media_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Block Media Framework Interface

#include <sf_block_media_api.h>

Data Fields

uint32_t	block_size Block size in bytes.
void const *	p_extend Instance dependent configuration.

Detailed Description

Interface Configuration

The documentation for this struct was generated from the following file:

- sf_block_media_api.h

sf_block_media_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Block Media Framework Interface

#include <sf_block_media_api.h>

Data Fields

ssp_err_t(*	open)(sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)
ssp_err_t(*	read)(sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t(*	write)(sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t(*	ioctl)(sf_block_media_ctrl_t *p_ctrl, ssp_command_t const command, void *p_data)
ssp_err_t(*	close)(sf_block_media_ctrl_t *p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for Block Media

Field Documentation

◆ close

ssp_err_t(* sf_block_media_api_t::close) (sf_block_media_ctrl_t *p_ctrl)

Close the open media channel.

Implemented as

- SF_Block_Media_SDMMC_Close()
- SF_BLOCK_MEDIA_RAM_Close()
- SF_BLOCK_MEDIA_QSPI_Close()
- SF_BLOCK_MEDIA_LX_NOR_Close()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.

◆ ioctl

ssp_err_t(* sf_block_media_api_t::ioctl) (sf_block_media_ctrl_t *p_ctrl, ssp_command_t const command, void *p_data)

Send control commands to and receives the status from the media port.

Implemented as

- SF_Block_Media_SDMMC_Control()
- SF_BLOCK_MEDIA_RAM_Control()
- SF_BLOCK_MEDIA_QSPI_Control()
- SF_BLOCK_MEDIA_LX_NOR_Control()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.
[in]	command	Command to execute.
[in,out]	p_data	Void pointer to data in or out.

◆ open

ssp_err_t(* sf_block_media_api_t::open) (sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)

Open a device channel for read/write and control.

Implemented as

- SF_Block_Media_SDMMC_Open()
- SF_BLOCK_MEDIA_RAM_Open()
- SF_BLOCK_MEDIA_QSPI_Open()
- SF_BLOCK_MEDIA_LX_NOR_Open()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.

◆ read

ssp_err_t(* sf_block_media_api_t::read) (sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

Read data from a media channel.

Implemented as

- SF_Block_Media_SDMMC_Read()
- SF_BLOCK_MEDIA_RAM_Read()
- SF_BLOCK_MEDIA_QSPI_Read()
- SF_BLOCK_MEDIA_LX_NOR_Read()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.
[in]	p_dest	Destination address to read data out.
[in]	start_sector	Beginning sector address to read.
[in]	sector_count	Number of sectors to read.

◆ versionGet

ssp_err_t(* sf_block_media_api_t::versionGet) (ssp_version_t *const p_version)

Return the version of the driver.

Implemented as

- SF_Block_Media_SDMMC_VersionGet()
- SF_BLOCK_MEDIA_RAM_VersionGet()
- SF_BLOCK_MEDIA_QSPI_VersionGet()
- SF_BLOCK_MEDIA_LX_NOR_VersionGet()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.
[out]	p_version	Memory address to return version information to.

◆ write

ssp_err_t(* sf_block_media_api_t::write) (sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)

Write data to a media channel.

Implemented as

- SF_Block_Media_SDMMC_Write()
- SF_BLOCK_MEDIA_RAM_Write()
- SF_BLOCK_MEDIA_QSPI_Write()
- SF_BLOCK_MEDIA_LX_NOR_Write()

Parameters

[in]	p_cfg	Pointer to the media configuration structure for a channel.
[in]	p_src	Source address of data for writing.
[in]	start_sector	Beginning sector address to write to.
[in]	sector_count	Number of sectors to write.

The documentation for this struct was generated from the following file:

- sf_block_media_api.h

sf_block_media_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Block Media Framework Interface

#include <sf_block_media_api.h>

Data Fields

sf_block_media_ctrl_t *	p_ctrl Block media pointer to device driver control structure.
sf_block_media_cfg_t const *	p_cfg

	Block media pointer to device driver configuration structure.
sf_block_media_api_t const *	p_api Block media pointer to device driver api structure.

Detailed Description

Interface Instance

The documentation for this struct was generated from the following file:

- sf_block_media_api.h

5.1.2.23 SF CELLULAR Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF CELUULAR Framework Interface. More...

Data Structures

struct	sf_cellular_provisioning_t
struct	sf_cellular_cmd_resp_t
struct	sf_cellular_ctrl_t
struct	sf_cellular_stats_t
struct	sf_cellular_info_t
struct	sf_cellular_network_status_t
struct	sf_cellular_command_parameters_info_t
struct	sf_cellular_callback_args_t
struct	sf_cellular_op_t
struct	sf_cellular_at_cmd_set_t
struct	sf_cellular_sim_pin_info_t

struct	sf_cellular_cfg_t
struct	sf_cellular_api_t
struct	sf_cellular_instance_t

Macros

#define	SF_CELLULAR_API_VERSION_MAJOR (2U)
#define	SF_CELLULAR_API_VERSION_MINOR (0U)
#define	SF_CELLULAR_ACESS_TECH_NAME_LEN (64U) Maximum length for Access Technology name.
#define	SF_CELLULAR_MAX_OPERATOR_NAME_LEN (32U) Maximum length for Operator name.
#define	SF_CELLULAR_MAX_PREFFERED_OPERATOR_COUNT (5U) Maximum number of preferred operator.
#define	SF_CELLULAR_IMEI_LEN (32U) Maximum length of IMEI number.
#define	SF_CELLULAR_FWVERSION_LEN (32U) Maximum length of Firmware Version.
#define	SF_CELLULAR_MAX_STRING_LEN (32U) Maximum string length.
#define	SF_CELLULAR_CHIPSET_LEN (16U) Maximum length of Chipset info.
#define	SF_CELLULAR_MFG_NAME_LEN (16U) Maximum length of manufacturer.
#define	SF_CELLULAR_CID_LEN (16U)

	Maximum length of CID.
#define	SF_CELLULAR_IMSI_LEN (24U) Maximum length of IMSI.
#define	SF_CELLULAR_IP_ADDR_LEN (128U) Maximum lenfth of IP address.
#define	SF_CELLULAR_TRUE (1U)
#define	SF_CELLULAR_FALSE (0U)

Enumerations

enum	sf_cellular_op_select_mode_t { SF_CELLULAR_OP_SELECT_MODE_AUTO = 0, SF_CELLULAR_OP_SELECT_MODE_MANUAL = 1, SF_CELLULAR_OP_SELECT_MODE_DEREGISTER = 2, SF_CELLULAR_OP_SELECT_MODE_MANUAL_FALLBACK = 4 }

enum	sf_cellular_network_reg_status_t {
	SF_CELLULAR_NETWORK_REG_STATUS_NOT_REGISTERED_NO_SEARCH = 0,
	SF_CELLULAR_NETWORK_REG_STATUS_REGISTERED_HOME_NETWORK = 1,
	SF_CELLULAR_NETWORK_REG_STATUS_NOT_REGISTERED_SEARCHING = 2,
	SF_CELLULAR_NETWORK_REG_STATUS_REGISTRATION_DENIED = 3,
	SF_CELLULAR_NETWORK_REG_STATUS_UNKNOWN = 4,
	SF_CELLULAR_NETWORK_REG_STATUS_REGISTERED_ROAMING = 5
	}

enum	sf_cellular_timezone_update_mode_t { SF_CELLULAR_TIMEZONE_UPDATE_AUTO_DISABLE, SF_CELLULAR_TIMEZONE_UPDATE_AUTO_ENABLE }
enum	sf_cellular_event_t { SF_CELLULAR_EVENT_RX, SF_CELLULAR_EVENT_PROVISIONSET }
enum	sf_cellular_reset_type_t { SF_CELLULAR_RESET_TYPE_SOFT, SF_CELLULAR_RESET_TYPE_HARD }
enum	sf_cellular_airplane_mode_t { SF_CELLULAR_AIRPLANE_MODE_OFF, SF_CELLULAR_AIRPLANE_MODE_ON }
enum	sf_cellular_pdp_type_t { SF_CELLULAR_PDP_TYPE_IP, SF_CELLULAR_PDP_TYPE_PPP, SF_CELLULAR_PDP_TYPE_IPV6,

	SF_CELLULAR_PDP_TYPE_IPV4V6 }
enum	sf_cellular_op_name_format_t { SF_CELLULAR_OP_NAME_FORMAT_LONG, SF_CELLULAR_OP_NAME_FORMAT_SHORT, SF_CELLULAR_OP_NAME_FORMAT_NUMERIC }
enum	sf_cellular_auth_type_t { SF_CELLULAR_AUTH_TYPE_NONE, SF_CELLULAR_AUTH_TYPE_PAP, SF_CELLULAR_AUTH_TYPE_CHAP }
enum	sf_cellular_at_cmd_index_t { SF_CELLULAR_AT_CMD_INDEX_AT = 0, SF_CELLULAR_AT_CMD_INDEX_ATZ0, SF_CELLULAR_AT_CMD_INDEX_AT_CREG_SET_0, SF_CELLULAR_AT_CMD_INDEX_AT_CMEE_SET_0, SF_CELLULAR_AT_CMD_INDEX_AT_ECHO, SF_CELLULAR_AT_CMD_INDEX_AT_SAVE, SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_CPIN, SF_CELLULAR_AT_CMD_INDEX_AT_CPIN_SET, SF_CELLULAR_AT_CMD_INDEX_AT_CGDCONT_SET, SF_CELLULAR_AT_CMD_INDEX_AT_CPOL_SET, SF_CELLULAR_AT_CMD_INDEX_AT_COPS_GET, SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_OFF, SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_ON, SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_ACTIVE, SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_DEACTIVE, SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_ACTIVE, SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_DEACTIVE, SF_CELLULAR_AT_CMD_INDEX_AT_CSQ_GET, SF_CELLULAR_AT_CMD_INDEX_AT_VER_GET, SF_CELLULAR_AT_CMD_INDEX_AT_CHIPSET_GET, SF_CELLULAR_AT_CMD_INDEX_AT_IMEI_GET, SF_CELLULAR_AT_CMD_INDEX_AT_MANF_NAME_GET, SF_CELLULAR_AT_CMD_INDEX_AT_SIMID_GET, SF_CELLULAR_AT_CMD_INDEX_AT_NET_TYPE_STATUS_GET, SF_CELLULAR_AT_CMD_INDEX_AT_NET_STATUS_GET, SF_CELLULAR_AT_CMD_INDEX_AT_DETACH, SF_CELLULAR_AT_CMD_INDEX_AT_LOCK_SIM, SF_CELLULAR_AT_CMD_INDEX_AT_UNLOCK_SIM, SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_DATA_MODE, SF_CELLULAR_AT_CMD_INDEX_AT_EXIT_DATA_MODE, SF_CELLULAR_AT_CMD_INDEX_AT_USERNAME_SET, SF_CELLULAR_AT_CMD_INDEX_AT_PASSWORD_SET, SF_CELLULAR_AT_CMD_INDEX_AT_AUTH_TYPE_SET, SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_ENABLE_SET, SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_DISABLE_SET , SF_CELLULAR_AT_CMD_INDEX_AT_EMPTY_APN_SET, SF_CELLULAR_AT_CMD_INDEX_AT_GET_IP_ADDR, SF_CELLULAR_AT_CMD_INDEX_AT_NWSCANSEQ_SET, SF_CELLULAR_AT_CMD_INDEX_AT_NWSCANMODE_SET, SF_CELLULAR_AT_CMD_INDEX_AT_IOTOPMODE_SET, SF_CELLULAR_AT_CMD_INDEX_AT_SWITCH_BACK_TO_DATA_MODE, SF_CELLULAR_AT_CMD_INDEX_AT_GET_IMSI, SF_CELLULAR_AT_CMD_INDEX_AT_IPR_SET,

	SF_CELLULAR_AT_CMD_INDEX_AT_BAUD_CHECK, SF_CELLULAR_AT_CMD_INDEX_AT_SIM_EFFECT_SET }

enum	sf_cellular_config_at_cmd_index_t {
	SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CREG,
	SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CEREG,
	SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_COPS_AUTO_SET,
	SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_COPS_MANUAL_SET,
	SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CPIN_STATUS_GET
	}

enum	sf_cellular_sim_status_t { SF_CELLULAR_SIM_STATUS_READY, SF_CELLULAR_SIM_STATUS_PIN_REQUIRED, SF_CELLULAR_SIM_STATUS_PIN_PUK_REQUIRED }
enum	sf_cellular_log_buffer_type_t { SF_CELLULAR_LOG_BUFFER_TYPE_COMMAND, SF_CELLULAR_LOG_BUFFER_TYPE_RESPONSE_RECEIVED, SF_CELLULAR_LOG_BUFFER_TYPE_RESPONSE_RECEIVED_WITH_TIMEOUT }

Detailed Description

RTOS-integrated SF CELUULAR Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF CELLULAR Framework.

Macro Definition Documentation

◆ SF_CELLULAR_API_VERSION_MAJOR

#define SF_CELLULAR_API_VERSION_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_CELLULAR_API_VERSION_MINOR

#define SF_CELLULAR_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

◆ SF_CELLULAR_FALSE

#define SF_CELLULAR_FALSE (0U)
Logical Value FALSE for Cellular

◆ SF_CELLULAR_TRUE

#define SF_CELLULAR_TRUE (1U)
Logical Value TRUE for Cellular

Enumeration Type Documentation

◆ sf_cellular_airplane_mode_t

enum sf_cellular_airplane_mode_t	
Airplane mode	
Enumerator	
SF_CELLULAR_AIRPLANE_MODE_OFF	Airplane mode disabled.
SF_CELLULAR_AIRPLANE_MODE_ON	Airplane mode enabled.

◆ sf_cellular_at_cmd_index_t

enum sf_cellular_at_cmd_index_t	
Enumeration for AT command index	
Enumerator	
SF_CELLULAR_AT_CMD_INDEX_AT	Index for Command AT.
SF_CELLULAR_AT_CMD_INDEX_ATZ0	Index for Command ATZ0.
SF_CELLULAR_AT_CMD_INDEX_AT_CREG_SET_0	Index for Command to set AT+CREG.
SF_CELLULAR_AT_CMD_INDEX_AT_CMEE_SET_0	Index for Command to set AT+CMEE.
SF_CELLULAR_AT_CMD_INDEX_AT_ECHO	Index for Command ATE.
SF_CELLULAR_AT_CMD_INDEX_AT_SAVE	Index for Command AT&W.
SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_CPIN	Index for Command to unlock SIM.
SF_CELLULAR_AT_CMD_INDEX_AT_CPIN_SET	Index for Command to set SIM PIN.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDCONT_SET	Index for Command to set AT+CGDCOND.
SF_CELLULAR_AT_CMD_INDEX_AT_CPOL_SET	Index for Command to set AT+CPOL.
SF_CELLULAR_AT_CMD_INDEX_AT_COPS_GET	

	Index for Command to get current operator details.
SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_OFF	Index for Command to set Airplane mode OFF.
SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_ON	Index for Command to set Airplane mode ON.
SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_ACTIVE	Index for Command to activate context.
SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_DEACTIVE	Index for Command to deactivate context.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_ACTIVE	Index for Command to activate Data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_DEACTIVE	Index for Command to deactivate Data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_CSQ_GET	Index for Command to get signal quality.
SF_CELLULAR_AT_CMD_INDEX_AT_VER_GET	Index for Command to get Modem stack Version.
SF_CELLULAR_AT_CMD_INDEX_AT_CHIPSET_GET	Index for Command to get chipset details.
SF_CELLULAR_AT_CMD_INDEX_AT_IMEI_GET	Index for Command to get IMEI number.
SF_CELLULAR_AT_CMD_INDEX_AT_MANF_NAME_GET	Index for Command to get manufacturer name.
SF_CELLULAR_AT_CMD_INDEX_AT_SIMID_GET	Index for Command to get SIM Card ID.
SF_CELLULAR_AT_CMD_INDEX_AT_NET_TYPE_STATUS_GET	Index for Command to get network type information.
SF_CELLULAR_AT_CMD_INDEX_AT_NET_STATUS_GET	Index for Command to get network status information.
SF_CELLULAR_AT_CMD_INDEX_AT_DETACH	Index for Command to detach the MT.
SF_CELLULAR_AT_CMD_INDEX_AT_LOCK_SIM	Index for Command to lock SIM card.
SF_CELLULAR_AT_CMD_INDEX_AT_UNLOCK_SIM	Index for Command to unlock SIM card.
SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_DATA_MODE	Index for Command to enter data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_EXIT_DATA_MODE	Index for Command to exit data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_USERNAME_SET	Index for Command to set username.

SF_CELLULAR_AT_CMD_INDEX_AT_PASSWORD_SET	Index for Command to set password.
SF_CELLULAR_AT_CMD_INDEX_AT_AUTH_TYPE_SET	Index for Command to set authorisation type.
SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_ENABLE_SET	Index for Command to enable auto time update.
SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_DISABLE_SET	Index for Command to disable auto time update.
SF_CELLULAR_AT_CMD_INDEX_AT_EMPTY_APN_SET	Index for Command to set empty APN.
SF_CELLULAR_AT_CMD_INDEX_AT_GET_IP_ADDR	Index for Command to get IP address.
SF_CELLULAR_AT_CMD_INDEX_AT_NWSCANSEQ_SET	Index for Command to set network fallback sequence.
SF_CELLULAR_AT_CMD_INDEX_AT_NWSCANMODE_SET	Index for Command to set network scan mode.
SF_CELLULAR_AT_CMD_INDEX_AT_IOTOPMODE_SET	Index for Command to Configure Network Category to be Searched under LTE.
SF_CELLULAR_AT_CMD_INDEX_AT_SWITCH_BACK_TO_DATA_MODE	Index for Command to switch back to Data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_GET_IMSI	Index for Command to get IMSI ID.
SF_CELLULAR_AT_CMD_INDEX_AT_IPR_SET	Index for Command to set baud rate of modem.
SF_CELLULAR_AT_CMD_INDEX_AT_BAUD_CHECK	Index for Command to check whether modem is responding after baud update.
SF_CELLULAR_AT_CMD_INDEX_AT_SIM_EFFECT_SET	Index for Command to set SIM priority effect.

◆ sf_cellular_auth_type_t

enum sf_cellular_auth_type_t	
Cellular authentication type	
Enumerator	
SF_CELLULAR_AUTH_TYPE_NONE	No authentication.
SF_CELLULAR_AUTH_TYPE_PAP	PAP.
SF_CELLULAR_AUTH_TYPE_CHAP	CHAP.

◆ sf_cellular_config_at_cmd_index_t

enum sf_cellular_config_at_cmd_index_t	
Enumeration for configurable AT command index	
Enumerator	
SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CREG	Index for Command AT+CREG.
SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CEREG	Index for Command AT+CEREG.
SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_COPS_AUTO_SET	Index for Command to set AUTO AT+COPS.
SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_COPS_MANUAL_SET	Index for Command to set Manual AT+COPS.
SF_CELLULAR_CONFIG_AT_CMD_INDEX_AT_CPIN_STATUS_GET	Index for Command to get status of SIM lock.

◆ sf_cellular_event_t

enum sf_cellular_event_t	
Cellular Framework event codes	
Enumerator	
SF_CELLULAR_EVENT_RX	Packet received event.
SF_CELLULAR_EVENT_PROVISIONSET	Provisioning Set event.

◆ sf_cellular_log_buffer_type_t

enum sf_cellular_log_buffer_type_t	
Command Buffer type specifies the type of data in debug log buffer	
Enumerator	
SF_CELLULAR_LOG_BUFFER_TYPE_COMMAND	Data in logging buffer is an AT command.
SF_CELLULAR_LOG_BUFFER_TYPE_RESPONSE_RECEIVED	Data in logging buffer is complete response received from modem.
SF_CELLULAR_LOG_BUFFER_TYPE_RESPONSE_RECEIVED_WITH_TIMEOUT	Data in logging buffer is response received from modem with timeout.

◆ sf_cellular_network_reg_status_t

enum sf_cellular_network_reg_status_t	
Cellular Network Registration Status	
Enumerator	
SF_CELLULAR_NETWORK_REG_STATUS_NOT_REGISTERED_NO_SEARCH	not registered, MT is not currently searching a new operator to register to
SF_CELLULAR_NETWORK_REG_STATUS_REGISTERED_HOME_NETWORK	registered, home network
SF_CELLULAR_NETWORK_REG_STATUS_NOT_REGISTERED_SEARCHING	not registered, but MT is currently searching a new operator to register to
SF_CELLULAR_NETWORK_REG_STATUS_REGISTRATION_DENIED	registration denied
SF_CELLULAR_NETWORK_REG_STATUS_UNKNOWN	registration status unknown
SF_CELLULAR_NETWORK_REG_STATUS_REGISTERED_ROAMING	Registered, roaming.

◆ sf_cellular_op_name_format_t

enum sf_cellular_op_name_format_t	
Cellular operator name format	
Enumerator	
SF_CELLULAR_OP_NAME_FORMAT_LONG	Long alphanumeric.
SF_CELLULAR_OP_NAME_FORMAT_SHORT	Short alphanumeric.
SF_CELLULAR_OP_NAME_FORMAT_NUMERIC	Numeric.

◆ sf_cellular_op_select_mode_t

enum sf_cellular_op_select_mode_t	
Operator selection mode	
Enumerator	
SF_CELLULAR_OP_SELECT_MODE_AUTO	Automatic Operator selection.
SF_CELLULAR_OP_SELECT_MODE_MANUAL	Manual Operator selection.
SF_CELLULAR_OP_SELECT_MODE_DEREGISTER	De-register from the network.
SF_CELLULAR_OP_SELECT_MODE_MANUAL_FALLBACK	Manual with fallback to automatic.

◆ sf_cellular_pdp_type_t

enum sf_cellular_pdp_type_t	
PDP type	
Enumerator	
SF_CELLULAR_PDP_TYPE_IP	Internet protocol.
SF_CELLULAR_PDP_TYPE_PPP	Point to point protocol.
SF_CELLULAR_PDP_TYPE_IPV6	Internet protocol, version 6.
SF_CELLULAR_PDP_TYPE_IPV4V6	Virtual introduced to handle dual stack UE capability.

◆ sf_cellular_reset_type_t

enum sf_cellular_reset_type_t	
Cellular Module reset type	
Enumerator	
SF_CELLULAR_RESET_TYPE_SOFT	Soft reset module using AT command.
SF_CELLULAR_RESET_TYPE_HARD	Hard reset module by toggling Reset Pin.

◆ sf_cellular_sim_status_t

enum sf_cellular_sim_status_t	
Enumeration for SIM lock status	
Enumerator	
SF_CELLULAR_SIM_STATUS_READY	SIM in Cellular modem is not pending for any password.
SF_CELLULAR_SIM_STATUS_PIN_REQUIRED	SIM in Cellular modem is lock and waiting for (U)SIM PIN to be given.
SF_CELLULAR_SIM_STATUS_PIN_PUK_REQUIRED	SIM in Cellular modem is lock and waiting for (U)SIM PIN and PUK to be given.

◆ sf_cellular_timezone_update_mode_t

enum sf_cellular_timezone_update_mode_t	
Timezone update mode	
Enumerator	
SF_CELLULAR_TIMEZONE_UPDATE_AUTO_DISABLE	Disable automatic time zone update.
SF_CELLULAR_TIMEZONE_UPDATE_AUTO_ENABLE	Enable automatic time zone update.

sf_cellular_provisioning_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint8_t	apn [SF_CELLULAR_MAX_STRING_LEN] Access Point Name.
sf_cellular_auth_type_t	auth_type Authentication type: PAP/CHAP.
uint8_t	username [SF_CELLULAR_MAX_STRING_LEN] User name used for authentication.
uint8_t	password [SF_CELLULAR_MAX_STRING_LEN] Password used for authentication.
sf_cellular_airplane_mode_t	airplane_mode Airplane mode.
uint8_t	context_id Context ID to be used for connection.
sf_cellular_pdp_type_t	pdp_type PDP Type for Context.

Detailed Description

Cellular Provisioning information structure

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_cmd_resp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint8_t *	p_buff
uint32_t	buff_len

Detailed Description

Modem Command/Response structure used to send Custom AT command and to receive response for the same

Field Documentation

◆ buff_len

uint32_t sf_cellular_cmd_resp_t::buff_len
AT command/Response buffer length. In case of Response this is both in and out parameter. Input is the length of buffer pointed by p_buff and output is number of bytes of response copied by framework. In case of command it is length of command in p_buff

◆ p_buff

uint8_t* sf_cellular_cmd_resp_t::p_buff
AT command/Response buffer. In case of AT command it is input buffer in which user should pass custom command to be sent. In case of Response it is output buffer in which framework will fill the response

---

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

void *	p_driver_handle Stores information required by underlying Cellular device driver.

Detailed Description

Cellular Framework control structure

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_stats_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint32_t	rx_bytes Bytes received successfully.
uint32_t	tx_bytes Bytes transmitted successfully.
uint32_t	rx_err Bytes receive errors.
uint32_t	tx_err Bytes transmit errors.

Detailed Description

The statistic and error counters for this instance

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint8_t	mfg_name [SF_CELLULAR_MFG_NAME_LEN] Manufacturer name.
uint8_t	chipset [SF_CELLULAR_CHIPSET_LEN] Pointer to string showing Cellular chipset/driver information.
uint8_t	fw_version [SF_CELLULAR_FWVERSION_LEN] Cellular firmware version.
uint8_t	imei [SF_CELLULAR_IMEI_LEN] IMEI number.
uint16_t	rssi Received signal strength indication.
uint16_t	ber Bit rate error.
uint8_t	ip_addr [SF_CELLULAR_IP_ADDR_LEN] IP address.

Detailed Description

Cellular Driver Information

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_network_status_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint16_t	country_code Country code.
uint32_t	operator_code Operator code.
int16_t	rssi RSSI.
uint8_t	cid [SF_CELLULAR_CID_LEN] Cell ID.
uint8_t	imsi [SF_CELLULAR_IMSI_LEN] IMSI.
uint8_t	op_name [SF_CELLULAR_MAX_OPERATOR_NAME_LEN] Operator name.
uint8_t	access_tech_name [SF_CELLULAR_ACESS_TECH_NAME_LEN] Access Technology name.

uint8_t	service_domain Service Domain.
uint8_t	active_band Active Band.
sf_cellular_network_reg_status_t	reg_status Cellular network registration status.

Detailed Description

Network Status information

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_command_parameters_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint8_t	retry_count Count for which AT command will be retried.
uint16_t	retry_delay Delay between AT command retry.
sf_cellular_config_at_cmd_index_t	cmd_index configurable AT command index

Detailed Description

Callback Structure to configure AT command parameters from user

---

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

sf_cellular_event_t	event Event Code.
uint8_t *	p_data Pointer to received data.
uint32_t	length Receive Data Length.
void const *	p_context Context Provided to user during callback.

Detailed Description

Callback structure for Cellular driver to get the data receive notification

---

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_op_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

sf_cellular_op_name_format_t	op_name_format Cellular operator name format.
uint8_t	op_name [SF_CELLULAR_MAX_OPERATOR_NAME_LEN] Cellular operator name.

Detailed Description

Preferred operator selection structure

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_at_cmd_set_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

uint8_t *	p_cmd AT Command.
uint8_t *	p_success_resp Success response string.

uint16_t	max_resp_length Maximum length of expected response.
uint32_t	resp_wait_time AT command response wait time in milliseconds.
uint8_t	retry Retry count.
uint16_t	retry_delay Delay between AT command retry.

Detailed Description

Structure defining AT commands parameters

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_sim_pin_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

sf_cellular_sim_status_t	sim_status
uint8_t *	p_sim_pin SIM Pin is used to unlock the SIM if the SIM is locked for SIM PIN.
uint8_t *	p_sim_puk SIM PUK is used to unlock the SIM if the SIM is locked for SIM PUK.

Detailed Description

Callback Structure to read SIM Pin and PUK from user

Field Documentation

◆ sim_status

sf_cellular_sim_status_t sf_cellular_sim_pin_info_t::sim_status
SIM Pin type expected from user to enter, in case of PUK type, user has to fill SIM Pin and PUK both pins

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

sf_cellular_op_select_mode_t	op_select_mode Cellular Operator selection mode.
sf_cellular_op_t	op Cellular operator. Valid when operator selection mode is manual.
uint16_t	num_pref_ops Number of preferred operators in the pref_ops array.
sf_cellular_op_t	pref_ops [SF_CELLULAR_MAX_PREFFERED_OPERATOR_COUNT] Array of structures describing preferred operators.
sf_cellular_timezone_update	tz_upd_mode

_mode_t	TimeZone update mode policy.
uint8_t *	p_sim_pin SIM Pin.
uint8_t *	p_puk_pin PUK Pin.
ssp_err_t(*	p_prov_callback)(sf_cellular_callback_args_t *p_args)
void(*	p_recv_callback)(sf_cellular_callback_args_t *p_args)
ssp_err_t(*	p_read_sim_pin_info_callback)(sf_cellular_sim_pin_info_t *p_args)
ssp_err_t(*	p_cmd_param_callback)(sf_cellular_command_parameters_info_t **p_args, uint8_t *p_at_cmd_num)
void const *	p_context User defined context passed into callback function.
void const *	p_extend Instance specific configuration.
sf_cellular_at_cmd_set_t const *	p_cmd_set Instance specific command set.
sf_cellular_at_cmd_set_t *	p_modifiable_cmd_set Instance specific modifiable command set.

Detailed Description

Define the Cellular configuration parameters

Field Documentation

◆ p_cmd_param_callback

ssp_err_t(* sf_cellular_cfg_t::p_cmd_param_callback) (sf_cellular_command_parameters_info_t **p_args, uint8_t *p_at_cmd_num)
Pointer to callback function to configure AT command parameters like retry delay and retry count at runtime

◆ p_prov_callback

ssp_err_t(* sf_cellular_cfg_t::p_prov_callback) (sf_cellular_callback_args_t *p_args)
Pointer to provisioning callback function, used in NSAL

◆ p_read_sim_pin_info_callback

ssp_err_t(* sf_cellular_cfg_t::p_read_sim_pin_info_callback) (sf_cellular_sim_pin_info_t *p_args)
Pointer to callback function to configure the SIM properties at runtime

◆ p_recv_callback

void(* sf_cellular_cfg_t::p_recv_callback) (sf_cellular_callback_args_t *p_args)
This is the receive callback function used by NetX which will take a data packet from the Cellular module and hand it over to NetX for further processing.

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

ssp_err_t(*	open)(sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg) Initializes and enables the Cellular module. More...

ssp_err_t(*	close)(sf_cellular_ctrl_t *const p_ctrl) Disables the Cellular module. More...
ssp_err_t(*	provisioningGet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t *const p_cellular_provisioning) Pointer to function to Get the Cellular module provisioning information. More...
ssp_err_t(*	provisioningSet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning) Pointer to function to Set the Cellular module's provisioning information. More...
ssp_err_t(*	infoGet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t *const p_cellular_info) Reads the Cellular module's information. More...
ssp_err_t(*	statisticsGet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_stats_t *const p_cellular_device_stats) Returns statistics information of Cellular module. More...
ssp_err_t(*	transmit)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length) Passes packet buffer to PPP stack for transmission. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...
ssp_err_t(*	networkConnect)(sf_cellular_ctrl_t *const p_ctrl) Initiates the Data connection. More...
ssp_err_t(*	networkDisconnect)(sf_cellular_ctrl_t *const p_ctrl) Terminates the Data connection. More...
ssp_err_t(*	networkStatusGet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status) Get Network Status information. More...

ssp_err_t(*	simPinSet)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_old_pin, uint8_t *const p_new_pin) Set SIM Pin. More...
ssp_err_t(*	simLock)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin) Locks SIM. More...
ssp_err_t(*	simUnlock)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin) Unlocks SIM. More...
ssp_err_t(*	simIDGet)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *p_sim_id) Gets the SIM ID. More...
ssp_err_t(*	commandSend)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_cmd_resp_t *const p_input_at_command, sf_cellular_cmd_resp_t *const p_output, uint32_t const timeout) Send AT command directly to Cellular Modem. More...
ssp_err_t(*	fotaCheck)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck) Checks for Available Firmware upgrade. More...
ssp_err_t(*	fotaStart)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart) Starts the Firmware upgrade. More...
ssp_err_t(*	fotaStop)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop) Stops the Firmware upgrade. More...
ssp_err_t(*	reset)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_reset_type_t reset_type) Reset cellular module. This reset() API will only work when module is opened. More...

Detailed Description

Cellular Framework API structure.

Field Documentation

◆ close

ssp_err_t(* sf_cellular_api_t::close) (sf_cellular_ctrl_t *const p_ctrl)

Disables the Cellular module.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module. .

◆ commandSend

ssp_err_t(* sf_cellular_api_t::commandSend) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_cmd_resp_t *const p_input_at_command, sf_cellular_cmd_resp_t *const p_output, uint32_t const timeout)

Send AT command directly to Cellular Modem.

This API will send AT command provided by user to the Cellular Modem and will collect the response from the Modem and will send it back to the user. If Modem is in Data Mode when this API is called then Framework will first switch Modem to Command Mode, then send the AT command and collect the response and then switches the Modem back to Data Mode.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_input_at_command	Pointer to structure which contains Modem command to send
[in,out]	p_output	Pointer to buffer in which response will be sent to user, Also user will pass the size of the buffer which is pointed by p_output
[in]	timeout	Timeout for which framework will wait for response

◆ fotaCheck

ssp_err_t(* sf_cellular_api_t::fotaCheck) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck)

Checks for Available Firmware upgrade.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_fotacheck	Pointer to fota check specific data structure

◆ fotaStart

ssp_err_t(* sf_cellular_api_t::fotaStart) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart)

Starts the Firmware upgrade.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_fotastart	Pointer to fota start specific data structure

◆ fotaStop

ssp_err_t(* sf_cellular_api_t::fotaStop) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop)

Stops the Firmware upgrade.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_fotastop	Pointer to fota stop specific data structure

◆ infoGet

ssp_err_t(* sf_cellular_api_t::infoGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t *const p_cellular_info)

Reads the Cellular module's information.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[out]	p_cellular_info	Pointer to Cellular info structure.

◆ networkConnect

ssp_err_t(* sf_cellular_api_t::networkConnect) (sf_cellular_ctrl_t *const p_ctrl)

Initiates the Data connection.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.

◆ networkDisconnect

ssp_err_t(* sf_cellular_api_t::networkDisconnect) (sf_cellular_ctrl_t *const p_ctrl)

Terminates the Data connection.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.

◆ networkStatusGet

ssp_err_t(* sf_cellular_api_t::networkStatusGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status)

Get Network Status information.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[out]	p_network_status	Pointer to Network Status structure

◆ open

ssp_err_t(* sf_cellular_api_t::open) (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg)

Initializes and enables the Cellular module.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to Cellular configuration structure.

◆ provisioningGet

ssp_err_t(* sf_cellular_api_t::provisioningGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t *const p_cellular_provisioning)

Pointer to function to Get the Cellular module provisioning information.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[out]	p_cellular_provisioning	Pointer to Cellular provisioning structure.

◆ provisioningSet

ssp_err_t(* sf_cellular_api_t::provisioningSet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning)

Pointer to function to Set the Cellular module's provisioning information.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_cellular_provisioning	Pointer to Cellular provisioning structure.

◆ reset

ssp_err_t(* sf_cellular_api_t::reset) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_reset_type_t reset_type)

Reset cellular module. This reset() API will only work when module is opened.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	reset_type	Reset type

◆ simIDGet

ssp_err_t(* sf_cellular_api_t::simIDGet) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *p_sim_id)

Gets the SIM ID.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[out]	p_sim_id	SIM ID

◆ simLock

ssp_err_t(* sf_cellular_api_t::simLock) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin)

Locks SIM.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_pin	PIN number to lock the SIM

◆ simPinSet

ssp_err_t(* sf_cellular_api_t::simPinSet) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_old_pin, uint8_t *const p_new_pin)

Set SIM Pin.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_old_pin	Pointer to char array containing current 4 digit pin.
[in]	p_new_pin	Pointer to char array containing new 4 digit pin.

◆ simUnlock

ssp_err_t(* sf_cellular_api_t::simUnlock) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin)

Unlocks SIM.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_pin	PIN number to unlock the SIM

◆ statisticsGet

ssp_err_t(* sf_cellular_api_t::statisticsGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_stats_t *const p_cellular_device_stats)

Returns statistics information of Cellular module.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[out]	p_cellular_device_stats	Pointer to Cellular statistics information structure.

◆ transmit

ssp_err_t(* sf_cellular_api_t::transmit) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length)

Passes packet buffer to PPP stack for transmission.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_buf	Pointer to packet buffer to transmit
[in]	length	Length of packet buffer

◆ versionGet

ssp_err_t(* sf_cellular_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	pointer to memory location to return version number

---

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

sf_cellular_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR Framework Interface

#include <sf_cellular_api.h>

Data Fields

sf_cellular_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_cellular_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_cellular_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_cellular_api.h

5.1.2.24 SF CELLULAR NSAL Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF CELLULAR NSAL Framework Interface. More...

Data Structures

struct	sf_cellular_nsal_cfg_t

Functions

ssp_err_t	sf_cellular_deinit (NX_IP_DRIVER *driver_req_ptr, sf_cellular_instance_t const *p_cellular_instance,

	sf_cellular_nsal_cfg_t *p_cellular_nsal_cfg) De-initialize the Cellular NSAL, deletes PPP interface, disconnect the network and close cellular framework. More...

Detailed Description

RTOS-integrated SF CELLULAR NSAL Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF CELLULAR NSAL Framework.

Function Documentation

◆ sf_cellular_deinit()

ssp_err_t sf_cellular_deinit (NX_IP_DRIVER * *p_driver_req_ptr*, sf_cellular_instance_t const * *p_cellular_instance*, sf_cellular_nsal_cfg_t * *p_cellular_nsal_cfg*)

De-initialize the Cellular NSAL, deletes PPP interface, disconnect the network and close cellular framework.

Function Prototypes

Parameters

[in]	p_driver_req_ptr	Pointer to NetX IP driver
[in]	p_cellular_instance	Pointer to Cellular Framework instance
[in]	p_cellular_nsal_cfg	Pointer to Cellular NSAL configuration structure

Return values

SSP_SUCCESS	Successfully de-initialized the cellular NSAL connection
SSP_ERR_CELLULAR_FAILED	Failed to de-init Cellular Framework or terminate the network connection
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Wait for the PPP Link down event, LCP Terminate ACK should have been received

Disconnect Network

Close Cellular Driver

Return an error to NetX.

sf_cellular_nsal_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF CELLULAR NSAL Framework Interface

#include <sf_cellular_nsal_api.h>

Data Fields

uint8_t *	p_ppp_stack PPP Stack.
uint32_t	ppp_stack_size PPP Stack size.
UINT	priority PPP Thread Priority.
NX_PPP *	p_ppp NetX PPP Interface.
NX_IP *	p_ip NetX IP Interface.
NX_PACKET_POOL *	p_ppp_packet_pool NetX Packet pool for internal.
void(*	p_ppp_invalid_packet_cb)(NX_PACKET *p_packet_ptr) Callback handler for Invalid packet.
void(*	p_ppp_send_byte)(UCHAR byte) PPP Send byte callback function.
void(*	p_link_down_cb)(NX_PPP *p_ppp_ptr) PPP Link down notification callback. More...
void(*	p_link_up_cb)(NX_PPP *p_ppp_ptr) PPP Link up notification callback.
sf_cellular_auth_type_t	auth_type Authentication Type.
UINT(*	p_chap_get_challenge_cb)(CHAR *p_rand_value, CHAR *p_id, CHAR

	*p_name) Get challenge notification callback. More...
UINT(*	p_chap_get_responder_cb)(CHAR *p_system, CHAR *p_name, CHAR *p_secret) Get Responder notification callback.
UINT(*	p_chap_get_verify_cb)(CHAR *p_system, CHAR *p_name, CHAR *p_secret) Get Chap verification callback.
UINT(*	p_pap_generate_login)(CHAR *p_name, CHAR *p_password) PAP Authentication generate login callback. More...
UINT(*	p_pap_verify_login)(CHAR *p_name, CHAR *p_password) PAP authentication verification callback.
uint32_t	local_ip Local IP Address.
uint32_t	peer_ip Peer IP Address.
void const *	p_extend Instance specific configuration.

Detailed Description

Define the NSAL configuration parameters

Field Documentation

◆ p_chap_get_challenge_cb

UINT(* sf_cellular_nsal_cfg_t::p_chap_get_challenge_cb) (CHAR *p_rand_value, CHAR *p_id, CHAR *p_name)
Get challenge notification callback. CHAP Callback Function

◆ p_link_down_cb

void(* sf_cellular_nsal_cfg_t::p_link_down_cb) (NX_PPP *p_ppp_ptr)
PPP Link down notification callback. Link Notification callback function

◆ p_pap_generate_login

UINT(* sf_cellular_nsal_cfg_t::p_pap_generate_login) (CHAR *p_name, CHAR *p_password)
PAP Authentication generate login callback. PAP Callback Function

---

The documentation for this struct was generated from the following file:

- sf_cellular_nsal_api.h

5.1.2.25 SF Socket CELLULAR Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF Socket Cellular Framework Interface. More...

Data Structures

struct	in_addr
struct	sockaddr
struct	sockaddr_in
struct	sf_cellular_socket_ctrl_t
struct	sf_cellular_socket_cfg_t

struct	sf_cellular_socket_api_t
struct	sf_cellular_socket_instance_t

Macros

#define	SF_CELLULAR_SOCKET_API_VERSION_MAJOR (2U)
#define	SF_CELLULAR_SOCKET_API_VERSION_MINOR (0U)
#define	SF_CELLULAR_SOCKET_NO_OF_BIT_IN_BYTE (8U)

Typedefs

typedef int32_t	socklen_t

Functions

int	socket (int domain, int type, int protocol) This creates socket for communication. More...
int	close (int sockfd) This closes socket. More...
int	bind (int sockfd, const struct sockaddr *p_local_sock_addr, socklen_t addrlen) This binds socket to IP address. More...
int	listen (int sockfd, int backlog) This listens for connection on socket. More...
int	connect (int sockfd, const struct sockaddr *p_serv_addr, socklen_t addrlen) This connects with remote socket(stream socket). More...
int	accept (int sockfd, struct sockaddr *p_cliaddr, socklen_t *p_addrlen) This accepts connection from remote socket. More...
ssize_t	send (int sockfd, const void *p_buf, size_t length, int flags) This sends data over STREAM socket. More...

ssize_t	recv (int sockfd, void *p_buf, size_t length, int flags) This receives data over STREAM socket. More...
ssize_t	sendto (int sockfd, const void *p_buf, size_t length, int flags, const struct sockaddr *p_dest_addr, socklen_t addrlen) This sends data over DGRAM socket. More...
ssize_t	recvfrom (int sockfd, void *p_buf, size_t length, int flags, struct sockaddr *p_src_addr, socklen_t *p_addrlen) This receives data over DGRAM socket. More...
int	setsockopt (int sockfd, int level, int optname, const void *p_optval, socklen_t optlen) This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...
int	getsockopt (int sockfd, int level, int optname, void *p_optval, socklen_t *p_optlen) This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...
int	select (int nfds, fd_set *p_readfds, fd_set *p_writefds, fd_set *p_exceptfds, struct timeval *p_timeout) This waits for any activity on socket. More...

Detailed Description

RTOS-integrated SF Socket Cellular Framework Interface.

Summary

This SSP Interface provides access OnChip stack BSD Socket API.

Macro Definition Documentation

◆ SF_CELLULAR_SOCKET_API_VERSION_MAJOR

#define SF_CELLULAR_SOCKET_API_VERSION_MAJOR (2U)
SF Cellular Socket APIs Major Version

◆ SF_CELLULAR_SOCKET_API_VERSION_MINOR

#define SF_CELLULAR_SOCKET_API_VERSION_MINOR (0U)
SF Cellular Socket APIs Minor Version

◆ SF_CELLULAR_SOCKET_NO_OF_BIT_IN_BYTE

#define SF_CELLULAR_SOCKET_NO_OF_BIT_IN_BYTE (8U)
SF Cellular Number of Bits in a Byte

Typedef Documentation

◆ socklen_t

typedef int32_t socklen_t
Socket address Length

Function Documentation

◆ accept()

int accept (int *sockfd*, struct sockaddr * *p_cliaddr*, socklen_t * *p_addrlen*)

This accepts connection from remote socket.

Accept connection request from remote.

Parameters

[in]	sockfd	Local socket
[out]	p_cliaddr	Pointer to remote socket address which trying to connect
[out]	p_addrlen	Pointer to address length of client socket address

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network address
[out]	p_addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error accepting the connection
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Connection is received successfully.

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network address (Can be NULL)
[out]	p_addrlen	Size of Socket address structure(Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error accepting the connection or invalid input parameters

Returns

Otherwise Connection is received successfully.

◆ bind()

int bind (int *sockfd*, const struct sockaddr * *p_local_sock_addr*, socklen_t *addrlen*)

This binds socket to IP address.

Bind socket to interface which is identified by IP address

Parameters

[in]	sockfd	Local socket
[in]	p_local_sock_addr	Pointer to local socket address
[in]	addrlen	Size of sock address structure

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Binding socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is bound successfully.

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Binding socket failed or invalid input parameters

Returns

Otherwise Socket is bound successfully.

◆ close()

int close (int *sockfd*)

This closes socket.

API which closes socket

Parameters

[in]	sockfd	Local socket

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Closing socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is closed successfully.

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Closing socket failed or invalid input parameters

Returns

Otherwise Socket is closed successfully.

◆ connect()

int connect (int *sockfd*, const struct sockaddr * *p_serv_addr*, socklen_t *addrlen*)

This connects with remote socket(stream socket).

Establish TCP connection with remote socket

Parameters

[in]	sockfd	Local socket
[in]	p_serv_addr	Pointer to remote socket address

[in]	addrlen	Size of sock address structure

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error occurred.
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is connected successfully.

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error occurred or invalid input parameters

Returns

Otherwise Socket is connected successfully.

◆ getsockopt()

int getsockopt (int *sockfd*, int *level*, int *optname*, void * *p_optval*, socklen_t * *p_optlen*)

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Get Socket options.

Parameters

[in]	sockfd	Local socket
[in]	level	Sockets API level
[in]	optname	Option to be get
[out]	p_optval	Option value to be get
[in]	p_optlen	Length of option value

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error reading socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option read successfully.

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error reading socket option or invalid socket descriptor

Returns

Otherwise Socket option read successfully.

◆ listen()

int listen (int *sockfd*, int *backlog*)

This listens for connection on socket.

Listen for tcp connection. Set socket in listen mode for tcp connection.

Parameters

[in]	sockfd	Local socket
[in]	backlog	Max number of connection queue.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Set socket in Listen mode successfully.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode or invalid input parameters

Returns

Otherwise Set socket in Listen mode successfully.

◆ recv()

ssize_t recv (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*)

This receives data over STREAM socket.

Receive data from remote socket.

Parameters

[in]	sockfd	Local socket
[out]	p_buf	Pointer to data buffer where data will be received
[in]	length	Maximum length of data which can be received
[in]	flags	Socket flags

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to receive data.

Returns

Otherwise Number of Data bytes received successfully.

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[out]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to receive data or invalid input parameters

Returns

Otherwise Data received successfully.

◆ recvfrom()

ssize_t recvfrom (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*, struct sockaddr * *p_src_addr*, socklen_t * *p_addrlen*)

This receives data over DGRAM socket.

Receive data from remote socket.

Parameters

[in]	sockfd	Local socket
[out]	p_buf	Pointer to data buffer where data will be received
[in]	length	Maximum length of data which can be received
[in]	flags	Socket flag
[out]	p_src_addr	Pointer to remote socket address which has sent data
[out]	p_addrlen	Length of socket address structure

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address
[in]	p_addrlen	Size of Remote machine network

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error receiving data

Returns

Otherwise Numbers of data bytes received successfully.

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to

receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address (Can be NULL)
[in]	p_addrlen	Size of Remote machine network (Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error receiving data or invalid input parameters

Returns

Otherwise Numbers of data bytes received successfully.

◆ select()

int select (int *nfds*, fd_set * *p_readfds*, fd_set * *p_writefds*, fd_set * *p_exceptfds*, struct timeval * *p_timeout*)

This waits for any activity on socket.

Wait on a given socket for specified amount of time. In case of any activity e.g. arrival of packet it comes out of wait.

Parameters

[in]	nfds	Max fd
[in]	p_readfds	Pointer to fd_set to check whether data is available for read
[in]	p_writefds	Pointer to fd_set to check whether data is available for write
[in]	p_exceptfds	Pointer to fd_set to check whether exceptional condition occurred
[in]	p_timeout	Wait time in milliseconds

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at

the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set. If no descriptor is to be tested for writing, p_writefds should be NULL
[in]	p_exceptfds	Exceptional socket fd set. If no descriptor is to be tested for exceptions, p_exceptfds should be NULL
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Timeout occurred, no activity.

Returns

Otherwise Activity detected(Packet available).

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set - API does not used this parameter
[in]	p_exceptfds	Exceptional socket fd set - API does not used this parameter
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Timeout occurred, no activity or invalid socket descriptor

Returns

Otherwise Activity detected(Packet available).

◆ send()

ssize_t send (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*)

This sends data over STREAM socket.

Send data to remote socket.

Parameters

[in]	sockfd	Local socket
[in]	p_buf	Pointer to data buffer
[in]	length	Data buffer length
[in]	flags	Socket flags

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to send data.

Returns

Otherwise Number of Data bytes sent successfully.

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to send data or invalid input parameters

Returns

Otherwise Data sent successfully.

◆ sendto()

ssize_t sendto (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*, const struct sockaddr * *p_dest_addr*, socklen_t *addrlen*)

This sends data over DGRAM socket.

Send data to remote socket.

Parameters

[in]	sockfd	Local socket
[in]	p_buf	Pointer to data buffer to sent
[in]	length	Data buffer length
[in]	flags	Socket flag
[in]	p_dest_addr	Pointer to remote socket address where to send data
[in]	addrlen	Length of socket address structure

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data
[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error Sending data.

Returns

Otherwise Numbers of bytes sent successfully.

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data

[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error Sending data or invalid input parameters

Returns

Otherwise Numbers of bytes sent successfully.

◆ setsockopt()

int setsockopt (int *sockfd*, int *level*, int *optname*, const void * *p_optval*, socklen_t *optlen*)

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Set Socket options.

Parameters

[in]	sockfd	Local socket
[in]	level	Sockets API level
[in]	optname	Option to be set
[in]	p_optval	Option value to be set
[in]	optlen	Length of option value

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error setting socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option set successfully.

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error setting socket option or invalid socket descriptor

Returns

Otherwise Socket option set successfully.

◆ socket()

int socket (int *domain*, int *type*, int *protocol*)

This creates socket for communication.

API which creates socket

Parameters

[in]	domain	Socket family
[in]	type	Socket type
[in]	protocol	Protocol type

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket descriptor of newly created socket

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket created successfully

sockaddr_in Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR

Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

uint16_t	sin_family Internet Protocol (AF_INET)
uint16_t	sin_port Address port (16 bits)
struct in_addr	sin_addr Internet address (32 bits)
int8_t	sin_zero [8] Not used. More...

Detailed Description

Socket address, Internet style.

Field Documentation

◆ sin_zero

int8_t sockaddr_in::sin_zero
Not used. Not used structure member.

The documentation for this struct was generated from the following files:

- sf_cellular_socket_api.h
- sf_socket_api.h
- sf_wifi_qca4010_socket_api.h
- nx_bsd.h

sf_cellular_socket_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR Framework Interface

#include <sf_cellular_socket_api.h>

Data Fields

sf_cellular_instance_t *	p_lower_lvl_cellular low level cellular interface

Detailed Description

Socket Interface control structure

The documentation for this struct was generated from the following file:

- sf_cellular_socket_api.h

sf_cellular_socket_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR Framework Interface

#include <sf_cellular_socket_api.h>

Data Fields

sf_cellular_instance_t *	p_lower_lvl_cellular Pointer to SF on-chip stack instance.
void *	p_extend Extended configuration.

Detailed Description

Socket Interface configuration structure

The documentation for this struct was generated from the following file:

- sf_cellular_socket_api.h

sf_cellular_socket_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR Framework Interface

#include <sf_cellular_socket_api.h>

Data Fields

ssp_err_t(*	open)(sf_cellular_socket_ctrl_t *p_ctrl, sf_cellular_socket_cfg_t const *const p_cfg) Pointer to function which initializes the network interface for data transfers Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer. More...
ssp_err_t(*	close)(sf_cellular_socket_ctrl_t *const p_ctrl) Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...
ssp_err_t(*	ping)(sf_cellular_socket_ctrl_t *const p_ctrl, ULONG *p_ip_addr, uint32_t count, uint32_t interval_ms)

Detailed Description

Socket Interface API

Field Documentation

◆ close

ssp_err_t(* sf_cellular_socket_api_t::close) (sf_cellular_socket_ctrl_t *const p_ctrl)

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block for the Cellular module Socket interface.

◆ open

ssp_err_t(* sf_cellular_socket_api_t::open) (sf_cellular_socket_ctrl_t *p_ctrl, sf_cellular_socket_cfg_t const *const p_cfg)

Pointer to function which initializes the network interface for data transfers Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to the control block for the Cellular module Socket interface.
[in]	p_cfg	Pointer to Cellular Socket interface configuration structure.

◆ ping

ssp_err_t(* sf_cellular_socket_api_t::ping) (sf_cellular_socket_ctrl_t *const p_ctrl, ULONG *p_ip_addr, uint32_t count, uint32_t interval_ms)

Pointer to a function which is used to ping the IP address.

Parameters

[in]	p_ctrl	Pointer to the control block
[in]	p_ip_addr	Pointer to IP address to ping
[in]	count	Number of ping attempts
[in]	interval_ms	Interval between ping attempts

◆ versionGet

ssp_err_t(* sf_cellular_socket_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters**		
[out]	p_version	Pointer to SSP Version structure

The documentation for this struct was generated from the following file:

- sf_cellular_socket_api.h

sf_cellular_socket_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR Framework Interface

#include <sf_cellular_socket_api.h>

Data Fields

sf_cellular_socket_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_cellular_socket_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_cellular_socket_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_cellular_socket_api.h

5.1.2.26 Communications Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated communications Framework Interface. More...

Data Structures

struct	sf_comms_callback_args_t
struct	sf_comms_cfg_t
struct	sf_comms_api_t
struct	sf_comms_instance_t

Macros

#define	SF_COMMS_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_comms_ctrl_t

Enumerations

enum	sf_comms_lock_t { SF_COMMS_LOCK_TX = 0, SF_COMMS_LOCK_RX, SF_COMMS_LOCK_ALL }
enum	sf_comms_event_t { SF_COMMS_DISCONNECT_EVENT }

Detailed Description

RTOS-integrated communications Framework Interface.

Implemented by:

- UART Framework Instance - UART implementation
- USB Communication Framework V2 - USBX CDC ACM device implementation
- Telnet Communication Framework on sf_comms_telnet - NetX telnet server with shared IP Instance implementation

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also Framework Communications Interface description: UART Communications Framework

Macro Definition Documentation

◆ SF_COMMS_API_VERSION_MAJOR

#define SF_COMMS_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_comms_ctrl_t

typedef void sf_comms_ctrl_t
Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. **Implemented as** ◦ sf_console_instance_ctrl_t

Enumeration Type Documentation

◆ sf_comms_event_t

enum sf_comms_event_t	
Options for the callback events. ***Note*** *Only applies for SF_COMMS_TELNET.*	
Enumerator	
SF_COMMS_DISCONNECT_EVENT	Disconnected the client.

◆ sf_comms_lock_t

enum sf_comms_lock_t	
Communications locks	
Enumerator	
SF_COMMS_LOCK_TX	Lock Transmit.
SF_COMMS_LOCK_RX	Lock Receive.
SF_COMMS_LOCK_ALL	Lock Transmit and Receive.

sf_comms_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Communications Framework Interface

#include <sf_comms_api.h>

Data Fields

sf_comms_event_t	event SF_COMMS callback event.

Detailed Description

sf_comms callback arguments definitions.

Note

Only applies for SF_COMMS_TELNET.

The documentation for this struct was generated from the following file:

- sf_comms_api.h

sf_comms_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Communications Framework Interface

#include <sf_comms_api.h>

Data Fields

void const *	p_extend Pointer to lower level communications control structure.

Detailed Description

Configuration for RTOS integrated communications driver

The documentation for this struct was generated from the following file:

- sf_comms_api.h

sf_comms_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Communications Framework Interface

#include <sf_comms_api.h>

Data Fields

ssp_err_t(*	open)(sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_comms_ctrl_t *const p_ctrl)
ssp_err_t(*	read)(sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)
ssp_err_t(*	write)(sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)
ssp_err_t(*	lock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)
ssp_err_t(*	unlock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Framework communications API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_comms_api_t::close) (sf_comms_ctrl_t *const p_ctrl)

Clean up communications driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.

◆ lock

ssp_err_t(* sf_comms_api_t::lock) (sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)

Lock the communications driver. Reserve exclusive access to the communications driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	lock_type	Locking type, transmission channel or reception channel
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ open

ssp_err_t(* sf_comms_api_t::open) (sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const p_cfg)

Initialize communications driver.

Parameters

[in,out]	p_ctrl	Pointer to a control structure allocated by user. The control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ read

ssp_err_t(* sf_comms_api_t::read) (sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)

Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	p_dest	Destination address to read data out
[in]	bytes	Read data length
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ unlock

ssp_err_t(* sf_comms_api_t::unlock) (sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)

Unlock the communications driver. Release exclusive access to the communications driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	lock_type	Locking type, transmission channel or reception channel

◆ versionGet

ssp_err_t(* sf_comms_api_t::versionGet) (ssp_version_t *const p_version)

Store the driver version in the provided p_version.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	p_version	Pointer to memory version to be stored.

◆ write

ssp_err_t(* sf_comms_api_t::write) (sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)
Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	p_src	Source address to read data out from
[in]	bytes	Write data length
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

The documentation for this struct was generated from the following file:

- sf_comms_api.h

sf_comms_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Communications Framework Interface

#include <sf_comms_api.h>

Data Fields

sf_comms_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_comms_cfg_t const *	p_cfg

	Pointer to the configuration structure for this instance.
sf_comms_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_comms_api.h

5.1.2.27 Console Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Console Framework Interface. More...

Data Structures

struct	sf_console_callback_args_t
struct	sf_console_command_t
struct	sf_console_menu_t
struct	sf_console_cfg_t
struct	sf_console_api_t
struct	sf_console_instance_t

Macros

#define	SF_CONSOLE_API_VERSION_MAJOR (2U)
#define	SF_CONSOLE_HELP_COMMAND ((uint8_t *) "?")
#define	SF_CONSOLE_MENU_PREVIOUS_COMMAND ((uint8_t *) "^")
#define	SF_CONSOLE_ROOT_MENU_COMMAND ((uint8_t *) "~")

#define	SF_CONSOLE_CALLBACK_NEXT_FUNCTION ((void(*)(sf_console_callback_args_t * p_args)) 0x70000000)

Typedefs

typedef void	sf_console_ctrl_t
typedef sf_console_callback_args_t	sf_console_cb_args_t

Detailed Description

RTOS-integrated Console Framework Interface.

Summary

This module is a ThreadX-aware Console Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also Console Interface description Console Framework

Macro Definition Documentation

◆ SF_CONSOLE_API_VERSION_MAJOR

#define SF_CONSOLE_API_VERSION_MAJOR (2U)
Version of the API defined in this file

◆ SF_CONSOLE_CALLBACK_NEXT_FUNCTION

#define SF_CONSOLE_CALLBACK_NEXT_FUNCTION ((void(*)(sf_console_callback_args_t * p_args)) 0x70000000)
Use this macro to access the next menu layer from this command.

◆ SF_CONSOLE_HELP_COMMAND

#define SF_CONSOLE_HELP_COMMAND ((uint8_t *) "?")
Command to print each command and help in menu

◆ SF_CONSOLE_MENU_PREVIOUS_COMMAND

#define SF_CONSOLE_MENU_PREVIOUS_COMMAND ((uint8_t *) "^")
Previous command

◆ SF_CONSOLE_ROOT_MENU_COMMAND

#define SF_CONSOLE_ROOT_MENU_COMMAND ((uint8_t *) "~")
Root menu command

Typedef Documentation

◆ sf_console_cb_args_t

typedef sf_console_callback_args_t sf_console_cb_args_t
DEPRECATED definition, please use sf_console_callback_args_t instead.

◆ sf_console_ctrl_t

typedef void sf_console_ctrl_t
Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. **Implemented as** ◦ sf_console_instance_ctrl_t

sf_console_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

sf_console_ctrl_t *	p_ctrl Pointer to console that received the command that caused this callback.

uint8_t const *	p_remaining_string String remaining after parsing command.
uint8_t const *	context Pointer to user provided data.
uint32_t	bytes The number of bytes remaining in the input string.

Detailed Description

Console callback arguments

The documentation for this struct was generated from the following file:

- sf_console_api.h

sf_console_command_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

uint8_t *	command Command string.
uint8_t *	help Description of command.
void(*	callback)(sf_console_callback_args_t *p_args) Callback to call when command is selected.
void const *	context

	User provided context passed into callback.

Detailed Description

Console command structure, used to create a console menu with associated callbacks.

The documentation for this struct was generated from the following file:

- sf_console_api.h

sf_console_menu_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

struct st_sf_console_menu const *	menu_prev Previous menu.
uint8_t const *	menu_name Menu name, used as a prompt.
uint32_t	num_commands Number of commands in this menu.
sf_console_command_t const *	command_list Pointer to an array of commands of length num_commands.

Detailed Description

Console menu structure.

The documentation for this struct was generated from the following file:

- sf_console_api.h

sf_console_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

sf_comms_instance_t const *	p_comms Pointer to communications driver instance.
sf_console_menu_t const *	p_initial_menu First menu to print during Open.
bool	echo Whether to echo input commands to transmitter.
bool	autostart If true, prompt will occur with p_initial_menu after initialization.

Detailed Description

Configuration for RTOS integrated console framework.

The documentation for this struct was generated from the following file:

- sf_console_api.h

sf_console_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

ssp_err_t(*	open)(sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg) This function configures the console. This function must be called before any other console functions. More...
ssp_err_t(*	close)(sf_console_ctrl_t *const p_ctrl) The close API handles cleans up internal driver data. More...
ssp_err_t(*	prompt)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout) Prints prompt string from menu, waits for input, parses input based on menu, and calls callback function if a command is identified. More...
ssp_err_t(*	parse)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes) Looks for input string in menu, and calls callback function if found. More...
ssp_err_t(*	read)(sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout) Reads data into the destination byte by byte and echos input to the console. Backspace, delete, and left/right arrow keys supported. Read completes when a line ending CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes input. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, read will return an error code. More...
ssp_err_t(*	write)(sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout) The write API gets mutex object and handles UART data transmission at UART HAL layer. gets event flag to synchronize to completion of data transfer. More...
ssp_err_t(*	argumentFind)(uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)

	Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Stores version information in provided pointer. More...

Detailed Description

Console framework API structure. Console implementations will use the following API.

Field Documentation

◆ argumentFind

ssp_err_t(* sf_console_api_t::argumentFind) (uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)

Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers.

Implemented as

- SF_CONSOLE_ArgumentFind()

Parameters

[in]	p_arg	Pointer to argument to find.
[in]	p_src	Pointer to source string to find the argument in.
[out]	p_index	Pointer to location to store index. Set to -1 if argument is not found in input string. Pass NULL if index is not requested.
[out]	p_data	Pointer to location to store data following the argument. Set to -1 if argument is not found in input string. Pass NULL if data is not requested.

◆ close

ssp_err_t(* sf_console_api_t::close) (sf_console_ctrl_t *const p_ctrl)

The close API handles cleans up internal driver data.

Implemented as

- SF_CONSOLE_Close()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for UART driver.

◆ open

ssp_err_t(* sf_console_api_t::open) (sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg)

This function configures the console. This function must be called before any other console functions.

Implemented as

- SF_CONSOLE_Open()

Parameters

[in,out]	p_ctrl	Pointer to a device structure allocated by user. The device control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ parse

ssp_err_t(* sf_console_api_t::parse) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)

Looks for input string in menu, and calls callback function if found.

Implemented as

- SF_CONSOLE_Parse()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for UART driver.
[in]	p_cmd_list	Pointer to a menu of valid input commands for this prompt
[in]	p_input	Pointer to a null terminated string to search for in the command list
[in]	bytes	Length of the input string.

◆ prompt

ssp_err_t(* sf_console_api_t::prompt) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout)

Prints prompt string from menu, waits for input, parses input based on menu, and calls callback function if a command is identified.

Implemented as

- SF_CONSOLE_Prompt()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for UART driver.
[in]	p_menu	Set to NULL to stay on current menu maintained by the console framework. To change menus, pass a pointer to the new menu.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ read

ssp_err_t(* sf_console_api_t::read) (sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout)

Reads data into the destination byte by byte and echos input to the console. Backspace, delete, and left/right arrow keys supported. Read completes when a line ending CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes input. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, read will return an error code.

Implemented as

- SF_CONSOLE_Read()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for UART driver.
[in]	p_dest	Destination address to read data out
[in]	bytes	Read data length
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ versionGet

ssp_err_t(* sf_console_api_t::versionGet) (ssp_version_t *const p_version)

Stores version information in provided pointer.

Implemented as

- SF_CONSOLE_VersionGet()

Parameters

[out]	p_version	Code and API version used stored here.

◆ write

ssp_err_t(* sf_console_api_t::write) (sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)
The write API gets mutex object and handles UART data transmission at UART HAL layer. gets event flag to synchronize to completion of data transfer. **Implemented as** ◦ SF_CONSOLE_Write() **Parameters**

[in]	p_ctrl	Pointer to device control block initialized in Open call for UART driver.
[in]	p_src	Pointer to a NULL terminated string. Length must be less than SF_CONSOLE_MAX_WRITE_LENGTH.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

The documentation for this struct was generated from the following file:

- sf_console_api.h

sf_console_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Console Framework Interface

#include <sf_console_api.h>

Data Fields

sf_console_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

sf_console_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_console_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_console_api.h

5.1.2.28 SSP Crypto Framework Common Module Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto Framework module. More...

Data Structures

struct	sf_crypto_data_handle_t
struct	sf_crypto_callback_args_t
struct	sf_crypto_cfg_t
struct	sf_crypto_api_t
struct	sf_crypto_instance_t

Macros

#define	SF_CRYPTO_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_crypto_ctrl_t

Enumerations

enum	sf_crypto_key_type_t {

	SF_CRYPTO_KEY_TYPE_RSA_PLAIN_TEXT, SF_CRYPTO_KEY_TYPE_RSA_CRT_PLAIN_TEXT, SF_CRYPTO_KEY_TYPE_RSA_WRAPPED, SF_CRYPTO_KEY_TYPE_AES_WRAPPED, SF_CRYPTO_KEY_TYPE_AES_PLAIN_TEXT, SF_CRYPTO_KEY_TYPE_ECC_PLAIN_TEXT, SF_CRYPTO_KEY_TYPE_ECC_WRAPPED, SF_CRYPTO_KEY_TYPE_ENCRYPTED_RSA_PRIVATE_KEY, SF_CRYPTO_KEY_TYPE_ENCRYPTED_AES_KEY, SF_CRYPTO_KEY_TYPE_ENCRYPTED_ECC_PRIVATE_KEY }
enum	sf_crypto_key_size_t { SF_CRYPTO_KEY_SIZE_RSA_1024, SF_CRYPTO_KEY_SIZE_RSA_2048, SF_CRYPTO_KEY_SIZE_AES_128, SF_CRYPTO_KEY_SIZE_AES_XTS_128, SF_CRYPTO_KEY_SIZE_AES_192, SF_CRYPTO_KEY_SIZE_AES_256, SF_CRYPTO_KEY_SIZE_AES_XTS_256, SF_CRYPTO_KEY_SIZE_ECC_192, SF_CRYPTO_KEY_SIZE_ECC_224, SF_CRYPTO_KEY_SIZE_ECC_256, SF_CRYPTO_KEY_SIZE_ECC_384 }
enum	sf_crypto_state_t { SF_CRYPTO_CLOSED, SF_CRYPTO_OPENED }
enum	sf_crypto_event_t { SF_CRYPTO_EVENT_PROCEDURE_DONE, SF_CRYPTO_EVENT_ERROR }
enum	sf_crypto_close_option_t { SF_CRYPTO_CLOSE_OPTION_DEFAULT, SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE }

Detailed Description

Interface definition for Synergy Crypto Framework module.

Summary

This is the Interface of SF_CRYPTO Framework module.

Crypto Common Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_API_VERSION_MAJOR

#define SF_CRYPTO_API_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework Common Module

Typedef Documentation

◆ sf_crypto_ctrl_t

typedef void sf_crypto_ctrl_t
SSP Crypto Framework Common Module control block. Allocate an instance specific control block to pass into the SSP Crypto Framework Common Module API calls. **Implemented as** ◦ sf_crypto_instance_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_close_option_t

enum sf_crypto_close_option_t	
SF_CRYPTO Close option. The module executes close operation if any SF_CRYPTO_XXX modules have already closed if SF_CRYPTO_CLOSE_OPTION_DEFAULT option is specified. The module performs close operation regardless of any SF_CRYPTO_XXX module status if SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE is specified.	
Enumerator	
SF_CRYPTO_CLOSE_OPTION_DEFAULT	Close the module if no any SF_CRYPTO_XXX modules opened.
SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE	Close the module regardless of SF_CRYPTO_XXX modules status.

◆ sf_crypto_event_t

enum sf_crypto_event_t	
Event code for the SSP Crypto Framework Common Module. This event code is all reserved for the future use.	
Enumerator	
SF_CRYPTO_EVENT_PROCEDURE_DONE	Crypto hardware procedure done.
SF_CRYPTO_EVENT_ERROR	Error occurred.

◆ sf_crypto_key_size_t

enum sf_crypto_key_size_t	
Supported key sizes	
Enumerator	
SF_CRYPTO_KEY_SIZE_RSA_1024	RSA 1024-bit key.
SF_CRYPTO_KEY_SIZE_RSA_2048	RSA 2048-bit key.
SF_CRYPTO_KEY_SIZE_AES_128	AES 128-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_XTS_128	AES 128-bit key for XTS chaining mode only.
SF_CRYPTO_KEY_SIZE_AES_192	AES 192-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_256	AES 256-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_XTS_256	AES 256-bit key for XTS chaining mode only.
SF_CRYPTO_KEY_SIZE_ECC_192	ECC 192-bit key.
SF_CRYPTO_KEY_SIZE_ECC_224	ECC 224-bit key.
SF_CRYPTO_KEY_SIZE_ECC_256	ECC 256-bit key.
SF_CRYPTO_KEY_SIZE_ECC_384	ECC 384-bit key.

◆ sf_crypto_key_type_t

enum sf_crypto_key_type_t	
Supported key types	
Enumerator	
SF_CRYPTO_KEY_TYPE_RSA_PLAIN_TEXT	RSA Key pair in standard format and plain text.
SF_CRYPTO_KEY_TYPE_RSA_CRT_PLAIN_TEXT	RSA Key pair in CRT format and plain text.
SF_CRYPTO_KEY_TYPE_RSA_WRAPPED	RSA Key pair public key in plain text and wrapped standard format private key.
SF_CRYPTO_KEY_TYPE_AES_WRAPPED	Wrapped AES key.
SF_CRYPTO_KEY_TYPE_AES_PLAIN_TEXT	AES Plain text key.
SF_CRYPTO_KEY_TYPE_ECC_PLAIN_TEXT	ECC Key pair in standard format and plain text.
SF_CRYPTO_KEY_TYPE_ECC_WRAPPED	ECC Key pair public key in plain text and wrapped standard format private key.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_RSA_PRIVATE_KEY	RSA private key in encrypted format for installation.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_AES_KEY	AES key in encrypted format for installation.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_ECC_PRIVATE_KEY	ECC private key in encrypted format for installation.

◆ sf_crypto_state_t

enum sf_crypto_state_t	
State codes for the SSP Crypto Framework Common Module	
Enumerator	
SF_CRYPTO_CLOSED	The module is closed.
SF_CRYPTO_OPENED	The module is opened.

sf_crypto_data_handle_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Framework Common Module Interface

#include <sf_crypto_api.h>

Data Fields

uint8_t *	p_data Pointer to data.
uint32_t	data_length The length of data pointed by p_data.

Detailed Description

A structure to handle data among Crypto Framework modules

The documentation for this struct was generated from the following file:

- sf_crypto_api.h

sf_crypto_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Framework Common Module Interface

#include <sf_crypto_api.h>

Data Fields

sf_crypto_event_t	event Event code of the low level hardware.
ssp_err_t	error Error code if SF_CRYPTO_EVENT_ERROR.

Detailed Description

Callback arguments for the SSP Crypto Framework Common Module

---

The documentation for this struct was generated from the following file:

- sf_crypto_api.h

sf_crypto_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Framework Common Module Interface

#include <sf_crypto_api.h>

Data Fields

uint32_t	wait_option Wait option for RTOS service calls.
crypto_instance_t *	p_lower_lvl_crypto Pointer to a low-level Crypto engine HAL driver instance.
void const *	p_extend Extension parameter for hardware specific settings.
void const *	p_context Placeholder for user data.
void *	p_memory_pool Byte pool address.
uint32_t	memory_pool_size Byte pool size.
sf_crypto_close_option_t	close_option Close option.

Detailed Description

Configuration structure for the SSP Crypto Framework Common Module

---

The documentation for this struct was generated from the following file:

- sf_crypto_api.h

sf_crypto_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Framework Common Module Interface

#include <sf_crypto_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_ctrl_t *const p_ctrl, sf_crypto_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_crypto_ctrl_t *const p_ctrl)
ssp_err_t(*	lock)(sf_crypto_ctrl_t *const p_ctrl)
ssp_err_t(*	unlock)(sf_crypto_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	statusGet)(sf_crypto_ctrl_t *const p_ctrl, sf_crypto_state_t *p_status)

Detailed Description

Shared Interface definition for the SSP Crypto Framework Common Module

Field Documentation

◆ close

ssp_err_t(* sf_crypto_api_t::close) (sf_crypto_ctrl_t *const p_ctrl)

Close SSP Crypto Framework Common Module. This function is to be called only once when the the Crypto services are no longer required.

Implemented as

- SF_CRYPTO_Close()

Parameters

[in,out]	p_ctrl	Pointer to a Crypto framework control block.

◆ lock

ssp_err_t(* sf_crypto_api_t::lock) (sf_crypto_ctrl_t *const p_ctrl)

Lock shared resources for Cryptography operations. This function is typically called by Crypto Framework modules (SF_CRYPTO_XXX) to protect shared software resources provided in Crypto Framework Common module or shared crypto hardware engine. Once lock() is called by a thread, any Crypto Framework services called by the other thread will be blocked until unlock() is called. The lock and unlock operations are managed by Crypto Framework modules so users do not need to call this function in typical use-cases. However, if this function is called by a user thread, users must be aware that any cryptography operations by the other threads will be locked out until unlock() is called by the thread which called lock().

Implemented as

- SF_CRYPTO_Lock()

Parameters

[in,out]	p_ctrl	Pointer to a Crypto framework control block.

◆ open

ssp_err_t(* sf_crypto_api_t::open) (sf_crypto_ctrl_t *const p_ctrl, sf_crypto_cfg_t const *const p_cfg)

Open SSP Crypto Framework Common Module. This function is to be called only once to initialize the Crypto services.

Implemented as

- SF_CRYPTO_Open()

Parameters

[in,out]	p_ctrl	Pointer to a Crypto framework control block. Must be declared by user.
[in]	p_cfg	Pointer to a Crypto framework configuration structure. All elements of this structure must be set by user.

◆ statusGet

ssp_err_t(* sf_crypto_api_t::statusGet) (sf_crypto_ctrl_t *const p_ctrl, sf_crypto_state_t *p_status)

Get status of SSP Crypto Framework Common Module.

Implemented as

- SF_CRYPTO_StatusGet()

Parameters

[in]	p_ctrl	Pointer to a Crypto framework control block.
[out]	p_status	Memory location to store module status.

◆ unlock

ssp_err_t(* sf_crypto_api_t::unlock) (sf_crypto_ctrl_t *const p_ctrl)

Unlock shared resources for Cryptography operations. This function is typically called by Crypto Framework modules (SF_CRYPTO_XXX) to allow any other threads to access to shared software resources provided in Crypto Framework Common module or shared crypto hardware engine. This function must be called by a thread which called lock(). The lock and unlock operations are managed by Crypto Framework modules so users do not need to call this function in typical use-cases. However, this function must be called by a user thread if the thread has ever called lock().

Implemented as

- SF_CRYPTO_Unlock()

Parameters

[in,out]	p_ctrl	Pointer to a Crypto framework control block.

◆ versionGet

ssp_err_t(* sf_crypto_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto Framework Common Module.

Implemented as

- SF_CRYPTO_VersionGet()

Parameters

[out]	p_version	Pointer to the memory to store the version information.

The documentation for this struct was generated from the following file:

- sf_crypto_api.h

sf_crypto_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Framework Common Module Interface

#include <sf_crypto_api.h>

Data Fields

sf_crypto_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_api.h

5.1.2.29 SSP Crypto Cipher Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto Cipher Framework module. More...

Data Structures

struct	sf_crypto_cipher_aes_init_params_t
struct	sf_crypto_cipher_rsa_init_params_t
struct	sf_crypto_cipher_cfg_t
struct	sf_crypto_cipher_api_t
struct	sf_crypto_cipher_instance_t

Typedefs

typedef void	sf_crypto_cipher_algorithm_init_params_t
typedef void	sf_crypto_cipher_ctrl_t

Enumerations

enum	sf_crypto_cipher_mode_t { SF_CRYPTO_CIPHER_MODE_ECB, SF_CRYPTO_CIPHER_MODE_CBC, SF_CRYPTO_CIPHER_MODE_CTR, SF_CRYPTO_CIPHER_MODE_XTS, SF_CRYPTO_CIPHER_MODE_GCM }
enum	sf_crypto_cipher_op_mode_t { SF_CRYPTO_CIPHER_OP_MODE_ENCRYPT, SF_CRYPTO_CIPHER_OP_MODE_DECRYPT }
enum	sf_crypto_cipher_padding_scheme_t { SF_CRYPTO_CIPHER_PADDING_SCHEME_NO_PADDING, SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS7, SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS1_1_5 }

Detailed Description

Interface definition for Synergy Crypto Cipher Framework module.

Summary

This is a ThreadX aware Interface of SF_CRYPTO_CIPHER Framework module which provides encryption and decryption operations for AES and RSA algorithms.

Crypto Cipher Framework Interface description: Crypto Framework

Typedef Documentation

◆ sf_crypto_cipher_algorithm_init_params_t

typedef void sf_crypto_cipher_algorithm_init_params_t

Algorithm specific parameters. Allocate an algorithm specific block to pass into the cipherInit API call.

Implemented as

- sf_crypto_cipher_aes_init_params_t for AES
- sf_crypto_cipher_rsa_init_params_t for RSA.

◆ sf_crypto_cipher_ctrl_t

typedef void sf_crypto_cipher_ctrl_t
SSP Crypto Cipher framework control block. Allocate an instance specific control block to pass into the SSP Crypto framework Cipher API calls. **Implemented as** ◦ sf_crypto_cipher_instance_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_cipher_mode_t

enum sf_crypto_cipher_mode_t	
AES modes for the SSP Crypto Cipher Framework	
Enumerator	
SF_CRYPTO_CIPHER_MODE_ECB	Electronic Code Book chaining mode, default for RSA.
SF_CRYPTO_CIPHER_MODE_CBC	Cipher Block Chaining.
SF_CRYPTO_CIPHER_MODE_CTR	Counter Mode.
SF_CRYPTO_CIPHER_MODE_XTS	XEX-based tweaked-codebook mode with ciphertext stealing.
SF_CRYPTO_CIPHER_MODE_GCM	Galois Counter Mode.

◆ sf_crypto_cipher_op_mode_t

enum sf_crypto_cipher_op_mode_t	
Operating mode for Cipher APIs	
Enumerator	
SF_CRYPTO_CIPHER_OP_MODE_ENCRYPT	The operating mode is set to encryption.
SF_CRYPTO_CIPHER_OP_MODE_DECRYPT	The operating mode is set to decryption.

◆ sf_crypto_cipher_padding_scheme_t

enum sf_crypto_cipher_padding_scheme_t	
Padding mode to be used for Cipher operation - encrypting/ decrypting input data	
Enumerator	
SF_CRYPTO_CIPHER_PADDING_SCHEME_NO_PADDING	No padding scheme.
SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS7	PKCS#7 padding scheme - applicable only for AES operations.
SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS1_1_5	PKCS#1 v1.5 padding scheme - applicable only for RSA operations.

sf_crypto_cipher_aes_init_params_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Cipher Framework Interface

#include <sf_crypto_cipher_api.h>

Data Fields

sf_crypto_data_handle_t *	p_iv pointer to IV for the AES operation.
sf_crypto_data_handle_t *	p_auth_tag

Detailed Description

AES Algorithm specific parameters for cipher operations

Field Documentation

◆ p_auth_tag

sf_crypto_data_handle_t* sf_crypto_cipher_aes_init_params_t::p_auth_tag
Pointer to the GCM Authentication Tag. Only tag length of SF_CRYPTO_CIPHER_AES_GCM_TAG_LENGTH_16_BYTES is supported.

The documentation for this struct was generated from the following file:

- sf_crypto_cipher_api.h

sf_crypto_cipher_rsa_init_params_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Cipher Framework Interface

#include <sf_crypto_cipher_api.h>

Detailed Description

RSA Algorithm specific parameters for cipher operations

The documentation for this struct was generated from the following file:

- sf_crypto_cipher_api.h

sf_crypto_cipher_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Cipher Framework Interface

#include <sf_crypto_cipher_api.h>

Data Fields

sf_crypto_key_type_t	key_type Key type for cipher operation.
sf_crypto_key_size_t	key_size Key size for cipher operation.
sf_crypto_cipher_mode_t	cipher_chaining_mode Chaining mode specified for the cipher operation.

sf_crypto_instance_t *	p_lower_lvl_crypto_common Pointer to a Crypto Framework common instance.
sf_crypto_trng_instance_t *	p_lower_lvl_crypto_trng Pointer to a Crypto Framework TRNG instance.
void const *	p_extend Future extension for hardware specific settings.

Detailed Description

Configuration structure for the SSP Crypto Cipher framework Cipher chaining mode for RSA operations is not applicable and can be set to ECB

The documentation for this struct was generated from the following file:

- sf_crypto_cipher_api.h

sf_crypto_cipher_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Cipher Framework Interface

#include <sf_crypto_cipher_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_cipher_cfg_t const *const p_cfg) Opens SSP Crypto Cipher framework. This function initializes a control block of the framework module based on the configuration parameters such as the key type, key size and chaining mode. The module allows users to have multiple instances with different control blocks, if required. More...
ssp_err_t(*	cipherInit)(sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const p_key, sf_crypto_cipher_algorithm_init_params_t *const p_algorithm_specific_params) Initializes a cipher operation. Must be called after open() or

	cipherFinal() is called, to initialize a new cipher operation. Unless a different key type or key size or chaining mode is used, users do not need to close the module for a new cipher operation but can call this function to restart another cipher operation. More...
ssp_err_t(*	cipherUpdate)(sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Encrypts / decrypts input data and writes it to the output buffer. Can be called multiple times for additional blocks of data. If input length is 0 this method does nothing. There may be 0 to (input length+block size - 1) bytes of data for AES operations. For RSA operation there will be no output until cipherFinal() is called. RSA Encryption is only supported with the RSA Public Key. RSA Decryption is only supported with the RSA Private Key. More...
ssp_err_t(*	cipherFinal)(sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Encrypts/decrypts all/last block of data and writes to the output buffer. Once cipherFinal() is called, no additional call of cipherUpdate() is allowed but cipherInit() can be called to initialize a new cipher operation unless another key type / key size/chaining mode is needed. In such a case a call to close() and open() is required. For AES operations, the number of bytes output into output data buffer may be larger or smaller than input length or even 0. RSA Encryption is only supported with the RSA Public Key. RSA Decryption is only supported with the RSA Private Key. More...
ssp_err_t(*	cipherAadUpdate)(sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_aad) Updates AAD (Additional Authenticated Data) for AES GCM operation. Can be called multiple times for additional blocks of data. This is ONLY to provide AAD for AES GCM operation. Not applicable to any other algorithms or modes. This has to be called prior to processing any plain text / cipher text data. In other words, before any call to cipherUpdate() or cipherFinal() is made. More...
ssp_err_t(*	close)(sf_crypto_cipher_ctrl_t *const p_ctrl) Closes SSP Crypto Cipher framework. This function resets a control block of the framework module and allows users to re-configure the module differently. For instance, users can close the module and re-open it with different key type or key size or chaining mode / algorithm for a new cipher operation. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Get version of SSP Crypto Cipher framework. More...

Detailed Description

Shared Interface definition for the SSP Crypto Cipher framework module

Field Documentation

◆ cipherAadUpdate

ssp_err_t(* sf_crypto_cipher_api_t::cipherAadUpdate) (sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_aad)

Updates AAD (Additional Authenticated Data) for AES GCM operation. Can be called multiple times for additional blocks of data. This is ONLY to provide AAD for AES GCM operation. Not applicable to any other algorithms or modes. This has to be called prior to processing any plain text / cipher text data. In other words, before any call to cipherUpdate() or cipherFinal() is made.

Implemented as

- SF_CRYPTO_CIPHER_CipherAadUpdate()

Parameters

[in]	p_ctrl	Pointer to Crypto Cipher Framework control block structure.
[in]	p_aad	Pointer to an input data buffer containing AAD and the AAD length.

Note

Data buffer must be WORD aligned.

◆ cipherFinal

ssp_err_t(* sf_crypto_cipher_api_t::cipherFinal) (sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out)

Encrypts/decrypts all/last block of data and writes to the output buffer. Once cipherFinal() is called, no additional call of cipherUpdate() is allowed but cipherInit() can be called to initialize a new cipher operation unless another key type / key size/chaining mode is needed. In such a case a call to close() and open() is required. For AES operations, the number of bytes output into output data buffer may be larger or smaller than input length or even 0. RSA Encryption is only supported with the RSA Public Key. RSA Decryption is only supported with the RSA Private Key.

Implemented as

- SF_CRYPTO_CIPHER_CipherFinal()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework control block structure.
[in]	p_data_in	Pointer to an input data buffer and the input data length.
[in,out]	p_data_out	Pointer to the output data buffer and the buffer size on input. If there is data to be output, buffer is filled and the length is updated.

Note

Data buffers must be WORD aligned.

◆ cipherInit

ssp_err_t(* sf_crypto_cipher_api_t::cipherInit) (sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const p_key, sf_crypto_cipher_algorithm_init_params_t *const p_algorithm_specific_params)

Initializes a cipher operation. Must be called after open() or cipherFinal() is called, to initialize a new cipher operation. Unless a different key type or key size or chaining mode is used, users do not need to close the module for a new cipher operation but can call this function to restart another cipher operation.

Implemented as

- SF_CRYPTO_CIPHER_CipherInit()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework control block structure.
[in]	cipher_operation_mode	Specifies encrypt or decrypt operation.
[in]	p_key	The key to be used for the cipher operation.
[in]	p_algorithm_specific_params	Algorithm specific parameters. Allocate and fill parameters specific to the algorithm for the key type configured at open().

◆ cipherUpdate

ssp_err_t(* sf_crypto_cipher_api_t::cipherUpdate) (sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out)

Encrypts / decrypts input data and writes it to the output buffer. Can be called multiple times for additional blocks of data. If input length is 0 this method does nothing. There may be 0 to (input length+block size - 1) bytes of data for AES operations. For RSA operation there will be no output until cipherFinal() is called. RSA Encryption is only supported with the RSA Public Key. RSA Decryption is only supported with the RSA Private Key.

Implemented as

- SF_CRYPTO_CIPHER_CipherUpdate()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework control block structure.
[in]	p_data_in	Pointer to an input data buffer and the input data length.
[in,out]	p_data_out	Pointer to an output data buffer and the buffer size on input. If there is data to be output, buffer is filled and the length is updated.

Note

Data buffers must be WORD aligned.

◆ close

ssp_err_t(* sf_crypto_cipher_api_t::close) (sf_crypto_cipher_ctrl_t *const p_ctrl)

Closes SSP Crypto Cipher framework. This function resets a control block of the framework module and allows users to re-configure the module differently. For instance, users can close the module and re-open it with different key type or key size or chaining mode / algorithm for a new cipher operation.

Implemented as

- SF_CRYPTO_CIPHER_Close()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework control block structure.

◆ open

ssp_err_t(* sf_crypto_cipher_api_t::open) (sf_crypto_cipher_ctrl_t *const p_ctrl, sf_crypto_cipher_cfg_t const *const p_cfg)

Opens SSP Crypto Cipher framework. This function initializes a control block of the framework module based on the configuration parameters such as the key type, key size and chaining mode. The module allows users to have multiple instances with different control blocks, if required.

Implemented as

- SF_CRYPTO_CIPHER_Open()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework control block structure. Caller only needs to allocate sf_crypto_cipher_instance_ctrl_t and not fill any parameters.
[in]	p_cfg	Pointer to sf_crypto_cipher_cfg_t configuration structure. All elements of this structure must be filled by caller.

◆ versionGet

ssp_err_t(* sf_crypto_cipher_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto Cipher framework.

Implemented as

- SF_CRYPTO_CIPHER_VersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_cipher_api.h

sf_crypto_cipher_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Cipher Framework Interface

#include <sf_crypto_cipher_api.h>

Data Fields

sf_crypto_cipher_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_cipher_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_cipher_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_cipher_api.h

5.1.2.30 SSP Crypto HASH Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto HASH Framework module. More...

Data Structures

struct	sf_crypto_hash_context_t
struct	sf_crypto_hash_callback_args_t
struct	sf_crypto_hash_cfg_t

struct	sf_crypto_hash_api_t
struct	sf_crypto_hash_instance_t

Macros

#define	SF_CRYPTO_HASH_API_VERSION_MAJOR (2U)
#define	SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_MD5 (16U) Message Digest size for SHA1. More...
#define	SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA1 (20U) Message Digest size for SHA1.
#define	SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA224 (28U) Message Digest size for SHA224.
#define	SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA256 (32U) Message Digest size for SHA256.

Typedefs

typedef void	sf_crypto_hash_ctrl_t

Enumerations

enum	sf_crypto_hash_state_t { SF_CRYPTO_HASH_CLOSED, SF_CRYPTO_HASH_OPENED, SF_CRYPTO_HASH_DIGEST_INITIALIZED, SF_CRYPTO_HASH_DIGEST_UPDATED }
enum	sf_crypto_hash_type_t { SF_CRYPTO_HASH_ALGORITHM_MD5, SF_CRYPTO_HASH_ALGORITHM_SHA1, SF_CRYPTO_HASH_ALGORITHM_SHA224, SF_CRYPTO_HASH_ALGORITHM_SHA256 }

Detailed Description

Interface definition for Synergy Crypto HASH Framework module.

Summary

This is the Interface of SF_CRYPTO_HASH Framework module.

Crypto HASH Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_HASH_API_VERSION_MAJOR

#define SF_CRYPTO_HASH_API_VERSION_MAJOR (2U)
The API version of SSP Crypto HASH Framework

◆ SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_MD5

#define SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_MD5 (16U)
Message Digest size for SHA1. Message Digest size for each HASH algorithm in bytes

Typedef Documentation

◆ sf_crypto_hash_ctrl_t

typedef void sf_crypto_hash_ctrl_t
SSP Crypto framework control block. Allocate an instance specific control block to pass into the SSP Crypto framework API calls. **Implemented as** ◦ sf_crypto_instance_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_hash_state_t

enum sf_crypto_hash_state_t	
State codes for the SSP SSP Crypto HASH Framework	
Enumerator	
SF_CRYPTO_HASH_CLOSED	The module is closed.
SF_CRYPTO_HASH_OPENED	The module is opened. The initial message digest is not yet generated.
SF_CRYPTO_HASH_DIGEST_INITIALIZED	Message digest is initialized.
SF_CRYPTO_HASH_DIGEST_UPDATED	Message digest is updated.

◆ sf_crypto_hash_type_t

enum sf_crypto_hash_type_t	
HASH algorithm types for the SSP SSP Crypto HASH Framework	
Enumerator	
SF_CRYPTO_HASH_ALGORITHM_MD5	MD5 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA1	SHA-1 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA224	SHA-224 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA256	SHA-256 algorithm type.

sf_crypto_hash_context_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto HASH Framework Interface

#include <sf_crypto_hash_api.h>

Data Fields

uint8_t *	p_message_digest Intermediate digest stored buffer - WORD aligned.
uint8_t *	p_message_digest_org Originally allocated buffer - may not be WORD aligned.
uint8_t *	p_message_buffer Intermediate message data stored buffer - - WORD aligned.
uint8_t *	p_message_buffer_org IOriginally allocated buffer - may not be WORD aligned.
uint64_t	message_bytes Number of bytes from user data processed.

uint32_t	message_bytes_buffered Number of bytes buffered in the message data stored buffer.

Detailed Description

HASH internal context structure for a message digest

The documentation for this struct was generated from the following file:

- sf_crypto_hash_api.h

sf_crypto_hash_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto HASH Framework Interface

#include <sf_crypto_hash_api.h>

Data Fields

ssp_err_t	error Error code if SF_CRYPTO_EVENT_ERROR.

Detailed Description

Callback arguments for the SSP Crypto HASH framework

The documentation for this struct was generated from the following file:

- sf_crypto_hash_api.h

sf_crypto_hash_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto HASH Framework Interface

#include <sf_crypto_hash_api.h>

Data Fields

sf_crypto_hash_type_t	hash_type HASH algorithm type.
sf_crypto_instance_t *	p_lower_lvl_crypto_common Pointer to a Crypto Framework common instance.
hash_instance_t *	p_lower_lvl_instance pointer to HASH lower-level module instance
void *	p_extend Pointer to an optional configuration for Crypto HAL module.

Detailed Description

Configuration structure for the SSP SSP Crypto HASH framework

---

The documentation for this struct was generated from the following file:

- sf_crypto_hash_api.h

sf_crypto_hash_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto HASH Framework Interface

#include <sf_crypto_hash_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_hash_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_crypto_hash_ctrl_t *const p_ctrl)
ssp_err_t(*	hashInit)(sf_crypto_hash_ctrl_t *const p_ctrl)
ssp_err_t(*	hashUpdate)(sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in)

ssp_err_t(*	hashFinal)(sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_data_handle_t *const p_msg_digest, uint32_t *p_size)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for the SSP SSP Crypto framework

Field Documentation

◆ close

ssp_err_t(* sf_crypto_hash_api_t::close) (sf_crypto_hash_ctrl_t *const p_ctrl)

Closes SSP Crypto HASH framework. This function de-initializes a control block of the framework module and allow users to re-configure the module differently. For instance, users can close the module and re-open it with different HASH algorithm for a new digest operation.

Implemented as

- SF_CRYPTO_HASH_Close()

Parameters

[in,out]	p_ctrl	Pointer to Crypto HASH Framework control block structure.

◆ hashFinal

ssp_err_t(* sf_crypto_hash_api_t::hashFinal) (sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_data_handle_t *const p_msg_digest, uint32_t *p_size)

Hashes the last block of data and returns a message digest in the output buffer. Once hashFinal() is called, no additional call of hashUpdate() is allowed but hashInit() can be called to initialize a new digest operation unless the other HASH algorithm type needed. If the other HASH algorithm is required for a new digest operation, call close() and open(). This is a blocking call.

Implemented as

- SF_CRYPTO_HASH_MessageDigestUpdate()

Parameters

[in]	p_ctrl	Pointer to Crypto HASH Framework control block structure.
[in,out]	p_msg_digest	Pointer to an output data buffer and the buffer size. Message digest will be generated in the buffer. Data buffer must be aligned to word alignment and the size must be sufficient to store the message digest.
[out]	p_size	Pointer to the 32-bit memory space to store the size of message digest.

◆ hashInit

ssp_err_t(* sf_crypto_hash_api_t::hashInit) (sf_crypto_hash_ctrl_t *const p_ctrl)

Initializes a message digest operation. Must be called once open() or hashFinal() is called to initialize a new digest operation. Unless a different HASH type is used, users do not need to close the module for a new digest operation but can call this function to restart another digest operation. This is a blocking call.

Implemented as

- SF_CRYPTO_HASH_MessageDigestInit()

Parameters

[in]	p_ctrl	Pointer to Crypto HASH Framework control block structure.

◆ hashUpdate

ssp_err_t(* sf_crypto_hash_api_t::hashUpdate) (sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in)

Hashes input data and saves it in an internal context buffer. Can be called multiple times for additional blocks of data. This is a blocking call.

Implemented as

- SF_CRYPTO_HASH_MessageDigestUpdate()

Parameters

[in]	p_ctrl	Pointer to Crypto HASH Framework control block structure.
[in]	p_data_in	Pointer to an input data buffer and the data length.

◆ open

ssp_err_t(* sf_crypto_hash_api_t::open) (sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_hash_cfg_t const *const p_cfg)

Opens SSP Crypto HASH framework. This function initializes a control block of the framework module based on the configuration parameters such as the HASH algorithm type. The module allows users to have multiple instances with different control blocks, if required.

Implemented as

- SF_CRYPTO_HASH_Open()

Parameters

[in,out]	p_ctrl	Pointer to Crypto HASH Framework control block structure.
[in]	p_cfg	Pointer to sf_crypto_hash_cfg_t configuration structure. All elements of this structure must be set by user.

◆ versionGet

ssp_err_t(* sf_crypto_hash_api_t::versionGet) (ssp_version_t *const p_version)
Get version of SSP Crypto HASH framework. **Implemented as** ◦ SF_CRYPTO_HASH_VersionGet() **Parameters** [in] p_version — Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_hash_api.h

sf_crypto_hash_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto HASH Framework Interface

#include <sf_crypto_hash_api.h>

Data Fields

sf_crypto_hash_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_hash_cfg_t *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_hash_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_hash_api.h

5.1.2.31 SSP Crypto Key Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto Key Framework module. More...

Data Structures

struct	sf_crypto_key_cfg_t
struct	sf_crypto_key_api_t
struct	sf_crypto_key_instance_t

Macros

#define	SF_CRYPTO_KEY_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_crypto_key_ctrl_t

Enumerations

enum	sf_crypto_key_state_t { SF_CRYPTO_KEY_CLOSED = 0, SF_CRYPTO_KEY_OPENED = 0x4F50454EU }

Detailed Description

Interface definition for Synergy Crypto Key Framework module.

Summary

This is the Interface of SF_CRYPTO_KEY Framework module. The Key framework module is a ThreadX aware Key Framework Interface which provides key generation services. This sits between the user application and HAL layer.

Crypto Key Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_KEY_API_VERSION_MAJOR

#define SF_CRYPTO_KEY_API_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework

Typedef Documentation

◆ sf_crypto_key_ctrl_t

typedef void sf_crypto_key_ctrl_t
SSP Crypto framework control block. Allocate an instance specific control block to pass into the SSP Crypto framework API calls. **Implemented as** ◦ sf_crypto_instance_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_key_state_t

enum sf_crypto_key_state_t	
State codes for the SSP Crypto Key framework module. Once the module is opened successfully, then the state is transition to OPENED state. After Key operations, the Key framework module must be closed with CLOSED state.	
Enumerator	
SF_CRYPTO_KEY_CLOSED	The Key module is closed.
SF_CRYPTO_KEY_OPENED	The Key module is opened The code means 'OPEN'.

sf_crypto_key_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Framework Interface

#include <sf_crypto_key_api.h>

Data Fields

sf_crypto_key_type_t	key_type Key type to be generated.

sf_crypto_key_size_t	key_size Key size to be generated.
sf_crypto_data_handle_t	domain_params
sf_crypto_data_handle_t	generator_point
sf_crypto_instance_t *	p_lower_lvl_crypto_common Pointer to a Crypto Framework common instance.
void const *	p_extend Extension parameter for hardware specific settings (Future purpose).

Detailed Description

Configuration structure for the SSP SSP Crypto Key framework

Field Documentation

◆ domain_params

sf_crypto_data_handle_t sf_crypto_key_cfg_t::domain_params
Pointer to domain parameters for the requested key type. Structure contains the domain data in the order a||b||p||n for ECC as defined in FIPS186-3 and data length. Length of the data to be in bytes. Should set to NULL for RSA and AES.

◆ generator_point

sf_crypto_data_handle_t sf_crypto_key_cfg_t::generator_point
Pointer to the generator base point of curve in the order Gx||Gy for ECC (where Gx and Gy are x and y coordinates respectively) and data length. Length of the data to be in bytes. This parameter applies only for ECC. For others (AES and RSA), this is UNUSED. Should set to NULL for RSA and AES.

The documentation for this struct was generated from the following file:

- sf_crypto_key_api.h

sf_crypto_key_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Framework Interface

#include <sf_crypto_key_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_crypto_key_ctrl_t *const p_ctrl)
ssp_err_t(*	keyGenerate)(sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_public_key)
ssp_err_t(*	EcdhSharedSecretCompute)(sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_t *const p_local_secret_key, sf_crypto_key_t *const p_remote_public_key, sf_crypto_key_t *const p_shared_secret)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for the SSP SSP Crypto framework

Field Documentation

◆ close

ssp_err_t(* sf_crypto_key_api_t::close) (sf_crypto_key_ctrl_t *const p_ctrl)

Close SSP Crypto Key framework.

Implemented as

- SF_CRYPTO_KEY_Close()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Key Framework control block structure.

◆ EcdhSharedSecretCompute

ssp_err_t(* sf_crypto_key_api_t::EcdhSharedSecretCompute) (sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_t *const p_local_secret_key, sf_crypto_key_t *const p_remote_public_key,

sf_crypto_key_t *const p_shared_secret)

Perform scalar multiplication for ECC algorithms only. This is a blocking call.

Implemented as

- SF_CRYPTO_KEY_EcdhSharedSecretCompute

Parameters

[in]	p_ctrl	Pointer to Crypto Key Framework control block structure.
[in]	p_local_secret_key	Pointer to a secret key structure. The pointer to the secret key and it's length in bytes, are to be populated on input. Refer to r_ecc_api.h for ECC key sizes. p_secret_key should be WORD aligned. The memory allocation to store the secret key is user's responsibility.
[in]	p_remote_public_key	pointer to a point on the curve data. The pointer to the point on curve data and its length in bytes, are to be populated on input. Refer to r_ecc_api.h for ECC point on curve sizes. p_point_on_curve should be WORD aligned. The memory allocation to store the point on curve data is user's responsibility.
[in,out]	p_shared_secret	The pointer to the buffer and it's length in bytes, are to be populated on input. On success the resultant point on curve data and it's length in bytes, are returned. Refer to r_ecc_api.h for ECC public key sizes. p_resultant_vector should be WORD aligned. The memory allocation to store the resultant point on curve data is user's responsibility.

◆ keyGenerate

ssp_err_t(* sf_crypto_key_api_t::keyGenerate) (sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_public_key)

Generate a key. This is a blocking call.

Implemented as

- SF_CRYPTO_KEY_Generate()

Parameters

[in]	p_ctrl	Pointer to Crypto Key Framework control block structure.
[in,out]	p_secret_key	Pointer to a secret key structure. The pointer to the buffer and it's length in bytes, are to be populated on input. On success the key and it's length in bytes, are returned. Refer to r_rsa_api.h for RSA secret key sizes. Refer to r_aes_api.h for AES key sizes. Refer to r_ecc_api.h for ECC key sizes. p_secret_key should be WORD aligned. The memory allocation to store the secret key is user's responsibility.
[in,out]	p_public_key	Pointer to a public key structure. The pointer to the buffer and it's length in bytes, are to be populated on input. On success the key and it's length in bytes, are returned. Refer to r_rsa_api.h for RSA public key sizes. Refer to r_ecc_api.h for ECC public key sizes. Should set to NULL for AES. p_public_key should be WORD aligned. The memory allocation to store the public key is user's responsibility.

◆ open

ssp_err_t(* sf_crypto_key_api_t::open) (sf_crypto_key_ctrl_t *const p_ctrl, sf_crypto_key_cfg_t const *const p_cfg)

Open SSP Crypto Key framework for subsequent call / Key generation.

Implemented as

- SF_CRYPTO_KEY_Open()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Key Framework control block structure.
[in]	p_cfg	Pointer to sf_crypto_key_cfg_t configuration structure. All elements of this structure must be set by user.

◆ versionGet

ssp_err_t(* sf_crypto_key_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto Key framework.

Implemented as

- SF_CRYPTO_KEY_VersionGet()

Parameters

[out]	p_version	Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_key_api.h

sf_crypto_key_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Framework Interface

#include <sf_crypto_key_api.h>

Data Fields

sf_crypto_key_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_key_cfg_t *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_key_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_key_api.h

5.1.2.32 SSP Crypto Key Installation Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto Key Installation Framework module. More...

Data Structures

struct	sf_crypto_key_installation_cfg_t
struct	sf_crypto_key_installation_api_t
struct	sf_crypto_key_installation_instance_t

Macros

#define	SF_CRYPTO_KEY_INSTALLATION_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_crypto_key_installation_ctrl_t

Enumerations

enum	sf_crypto_key_installation_state_t { SF_CRYPTO_KEY_INSTALLATION_CLOSED = 0, SF_CRYPTO_KEY_INSTALLATION_OPENED = 1 }
enum	sf_crypto_key_installation_shared_index_t { SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_0, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_1, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_2, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_3, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_4, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_5, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_6, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_7, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_8, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_9, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_A, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_B, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_C, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_D, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_E, SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_F }

Detailed Description

Interface definition for Synergy Crypto Key Installation Framework module.

Summary

This is the Interface of SF_CRYPTO_KEY_INSTALLATION Framework module. The Key Installation framework module is a ThreadX aware Framework Interface which provides key installation services. This sits between the user application and HAL layer.

Crypto Key Installation Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_KEY_INSTALLATION_API_VERSION_MAJOR

#define SF_CRYPTO_KEY_INSTALLATION_API_VERSION_MAJOR (2U)
The API version of SSP Crypto Key Installation Framework

Typedef Documentation

◆ sf_crypto_key_installation_ctrl_t

typedef void sf_crypto_key_installation_ctrl_t

SSP Crypto Key installation framework control block. Allocate an instance specific control block to pass into the SSP Crypto Key Installation framework API calls.

Implemented as

- sf_crypto_key_installation_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_key_installation_shared_index_t

enum sf_crypto_key_installation_shared_index_t	
Supported shared key index values	
Enumerator	
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_0	Shared Key Index 0.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_1	Shared Key Index 1.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_2	Shared Key Index 2.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_3	Shared Key Index 3.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_4	Shared Key Index 4.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_5	Shared Key Index 5.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_6	Shared Key Index 6.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_7	Shared Key Index 7.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_8	Shared Key Index 8.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_9	Shared Key Index 9.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_A	Shared Key Index 10.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_B	Shared Key Index 11.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_C	Shared Key Index 12.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_D	Shared Key Index 13.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_E	Shared Key Index 14.
SF_CRYPTO_KEY_INSTALLATION_SHARED_INDEX_F	Shared Key Index 15.

◆ sf_crypto_key_installation_state_t

enum sf_crypto_key_installation_state_t	
State codes for the SSP Crypto Key installation framework module. Once the module is opened successfully, then the state is transition to OPENED state. After Key Installation operations, the Key installation framework module must be closed with CLOSED state.	
Enumerator	
SF_CRYPTO_KEY_INSTALLATION_CLOSED	The Key Installation module is closed.
SF_CRYPTO_KEY_INSTALLATION_OPENED	The Key Installation module is opened.

sf_crypto_key_installation_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Installation Framework Interface

#include <sf_crypto_key_installation_api.h>

Data Fields

sf_crypto_key_type_t	key_type Type of key to be installed.
sf_crypto_key_size_t	key_size Size of key to be installed.
sf_crypto_instance_t *	p_lower_lvl_common Pointer to a Crypto Framework common instance.
key_installation_instance_t *	p_lower_lvl_instance Pointer to Crypto Key Install HAL instance.
void const *	p_extend Extension parameter for hardware specific settings (Future purpose).

Detailed Description

Configuration structure for the SSP SSP Crypto Key Installation framework

---

The documentation for this struct was generated from the following file:

- sf_crypto_key_installation_api.h

sf_crypto_key_installation_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Installation Framework Interface

#include <sf_crypto_key_installation_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_key_installation_ctrl_t *const p_ctrl, sf_crypto_key_installation_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_crypto_key_installation_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	keyInstall)(sf_crypto_key_installation_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_user_key_rsa_modulus, sf_crypto_data_handle_t const *const p_user_key_input, sf_crypto_key_installation_shared_index_t const shared_index_input, sf_crypto_data_handle_t const *const p_session_key_input, uint32_t const *const p_iv_input, sf_crypto_data_handle_t *const p_key_data_out)

Detailed Description

Shared Interface definition for the SSP Crypto Key Installation Interface framework

Field Documentation

◆ close

ssp_err_t(* sf_crypto_key_installation_api_t::close) (sf_crypto_key_installation_ctrl_t *const p_ctrl)

Close SSP Crypto Key Installation framework.

Implemented as

- SF_CRYPTO_KEY_INSTALLATION_Close()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Key Installation Framework control block structure.

◆ keyInstall

ssp_err_t(* sf_crypto_key_installation_api_t::keyInstall) (sf_crypto_key_installation_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_user_key_rsa_modulus, sf_crypto_data_handle_t const *const p_user_key_input, sf_crypto_key_installation_shared_index_t const shared_index_input, sf_crypto_data_handle_t const *const p_session_key_input, uint32_t const *const p_iv_input, sf_crypto_data_handle_t *const p_key_data_out)

Install a key from the user's encrypted key, a shared index, session key, and an IV generated using a scheme designed to maintain plaintext source key isolation. This returns a wrapped key (sometimes called a key index) that can be used in other crypto APIs in place of the associated plaintext key (stored offline).

Implemented as

- SF_CRYPTO_KEY_INSTALLATION_KeyInstall()

Parameters

[in]	p_ctrl	Pointer to Crypto Key Installation Framework control block structure. Caller should not modify any elements of this structure at any time.
[in]	p_user_key_rsa_modulus	Pointer to sf_crypto_key_handle_t structure which includes a pointer to the WORD aligned buffer which holds the RSA modulus portion of the encrypted user RSA private key and the modulus length. This is only applicable when a RSA standard key is being installed. To be set to NULL otherwise.
[in]	p_user_key_input	Pointer to

		sf_crypto_key_handle_t structure which includes a pointer to the WORD aligned buffer which holds the encrypted user key and length. This is the key to be installed in encrypted format.
[in]	shared_index_input	An enumerated type that reflects the shared key index returned by the DLM Service, accompanied by the session key that follows.
[in]	p_session_key_input	Pointer to sf_crypto_key_handle_t structure which includes a pointer to the WORD aligned buffer which holds the session key and length returned by the DLM Service, accompanied by the shared index key, above.
[in]	p_iv_input	Pointer to the 128-bit IV array used to encrypt p_user_key_input.
[in,out]	p_key_data_out	Pointer to sf_crypto_key_handle_t structure which includes a pointer to the WORD aligned buffer to hold the wrapped key and the buffer length. This is the wrapped key returned after key installation.

Note

It is the user's responsibility to ensure all the above input/output buffers are WORD aligned.
Caller must assign appropriate length to data_length field for all buffers before calling this API.

◆ open

ssp_err_t(* sf_crypto_key_installation_api_t::open) (sf_crypto_key_installation_ctrl_t *const p_ctrl, sf_crypto_key_installation_cfg_t const *const p_cfg)

Open SSP Crypto Key Installation framework for subsequent call / Key installation.

Implemented as

- SF_CRYPTO_KEY_INSTALLATION_Open()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Key Installation Framework control block structure.
[in]	p_cfg	Pointer to sf_crypto_key_installation_cfg_t configuration structure. All elements of this structure must be set by user.

◆ versionGet

ssp_err_t(* sf_crypto_key_installation_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto Key Installation framework.

Implemented as

- SF_CRYPTO_KEY_INSTALLATION_VersionGet()

Parameters

[out]	p_version	Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_key_installation_api.h

sf_crypto_key_installation_instance_t Struct Reference
Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Key Installation Framework Interface

#include <sf_crypto_key_installation_api.h>

Data Fields

sf_crypto_key_installation_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_key_installation_cfg_t *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_key_installation_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_key_installation_api.h

5.1.2.33 SSP Crypto Signature Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto Signature Framework module. More...

Data Structures

struct	sf_crypto_signature_rsa_specific_params_t
struct	sf_crypto_signature_cfg_t
struct	sf_crypto_signature_api_t
struct	sf_crypto_signature_instance_t

Macros

#define	SF_CRYPTO_SIGNATURE_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_crypto_signature_algorithm_init_params_t
typedef void	sf_crypto_signature_ctrl_t

Enumerations

enum	sf_crypto_signature_mode_t { SF_CRYPTO_SIGNATURE_MODE_SIGN, SF_CRYPTO_SIGNATURE_MODE_VERIFY }
enum	sf_crypto_signature_message_operation_t { SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_NONE, SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA1_PKCS1_1_5, SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA224_PKCS1_1_5, SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA256_PKCS1_1_5 }

Detailed Description

Interface definition for Synergy Crypto Signature Framework module.

Summary

The Signature framework module is a ThreadX aware module which provides sign and sign-verify services. They Key type and Key size provided in the configuration parameter determine the cryptography algorithm type and uses the appropriate Driver API interface to provide requested functionality. User can change the operation mode (Sign / Verify), message format and key data input multiple times after opening this module using the Open API. There is no need to close the module using Close API and then re-open the module if the intent is to just change operation mode, message format and/or key data input parameters.

Crypto Signature Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_SIGNATURE_API_VERSION_MAJOR

#define SF_CRYPTO_SIGNATURE_API_VERSION_MAJOR (2U)
The API version of SSP Crypto Signature Framework

Typedef Documentation

◆ sf_crypto_signature_algorithm_init_params_t

typedef void sf_crypto_signature_algorithm_init_params_t

Algorithm specific parameters. Allocate an algorithm specific block to pass into the contextInit API call.

Implemented as

- sf_crypto_signature_rsa_specific_params_t for RSA

◆ sf_crypto_signature_ctrl_t

typedef void sf_crypto_signature_ctrl_t

SSP Crypto Signature framework control block. Allocate an instance specific control block to pass into the SSP Crypto Signature framework API calls.

Implemented as

- sf_crypto_signature_instance_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_signature_message_operation_t

enum sf_crypto_signature_message_operation_t

Signature message operation

Enumerator	
SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_NONE	Input message is pre-formatted using appropriate format. Sign/verify operation is performed on the input message.
SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA1_PKCS1_1_5	Generates a 20-byte (SHA-1) digest and signs/verifies the digest using RSASSA-PKCS1 v1.5 padding scheme.
SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA224_PKCS1_1_5	Generates a 28-byte (SHA-224) digest and signs/verifies the digest using RSASSA-PKCS1 v1.5 padding scheme.
SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA256_PKCS1_1_5	Generates a 32-byte (SHA-256) digest and signs/verifies the digest using RSASSA-PKCS1 v1.5 padding scheme.

◆ sf_crypto_signature_mode_t

enum sf_crypto_signature_mode_t	
Signature mode, sign or verify	
Enumerator	
SF_CRYPTO_SIGNATURE_MODE_SIGN	Perform Sign Operation.
SF_CRYPTO_SIGNATURE_MODE_VERIFY	Perform Verify Operation.

sf_crypto_signature_rsa_specific_params_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Signature Framework Interface

#include <sf_crypto_signature_api.h>

Data Fields

sf_crypto_signature_message_operation_t	message_format

Detailed Description

RSA Algorithm specific parameters for signature operations

Field Documentation

◆ message_format

sf_crypto_signature_message_operation_t sf_crypto_signature_rsa_specific_params_t::message_format
Message format enumeration option.

The documentation for this struct was generated from the following file:

- sf_crypto_signature_api.h

sf_crypto_signature_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Signature Framework Interface

#include <sf_crypto_signature_api.h>

Data Fields

sf_crypto_key_type_t	key_type
sf_crypto_key_size_t	key_size
sf_crypto_hash_instance_t *	p_lower_lvl_sf_crypto_hash
sf_crypto_instance_t *	p_lower_lvl_crypto_common
void const *	p_extend

Detailed Description

Configuration structure for the SSP Crypto Signature framework

Field Documentation

◆ key_size

sf_crypto_key_size_t sf_crypto_signature_cfg_t::key_size
Key Size.

◆ key_type

sf_crypto_key_type_t sf_crypto_signature_cfg_t::key_type
Key Type.

◆ p_extend

void const* sf_crypto_signature_cfg_t::p_extend
Extension parameter for hardware specific settings (Future purpose).

◆ p_lower_lvl_crypto_common

sf_crypto_instance_t* sf_crypto_signature_cfg_t::p_lower_lvl_crypto_common
Pointer to a Crypto Framework common instance.

◆ p_lower_lvl_sf_crypto_hash

sf_crypto_hash_instance_t* sf_crypto_signature_cfg_t::p_lower_lvl_sf_crypto_hash
Pointer to Hash framework instance.

The documentation for this struct was generated from the following file:

- sf_crypto_signature_api.h

sf_crypto_signature_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Signature Framework Interface

#include <sf_crypto_signature_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_signature_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_crypto_signature_ctrl_t *const p_ctrl)
ssp_err_t(*	contextInit)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_signature_mode_t operation_mode, sf_crypto_signature_algorithm_init_params_t *const p_algorithm_specific_params, sf_crypto_key_t const *const p_key)
ssp_err_t(*	signUpdate)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message)
ssp_err_t(*	verifyUpdate)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message)
ssp_err_t(*	signFinal)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message, sf_crypto_data_handle_t *const p_dest)

ssp_err_t(*	verifyFinal)(sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_signature, sf_crypto_data_handle_t const *const p_message)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for the SSP Crypto Signature framework

Field Documentation

◆ close

ssp_err_t(* sf_crypto_signature_api_t::close) (sf_crypto_signature_ctrl_t *const p_ctrl)

Close SSP Crypto Signature framework module. This API will free any memory allocated when the signature framework module was opened.

Implemented as

- SF_CRYPTO_SIGNATURE_Close()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework control block structure.

◆ contextInit

ssp_err_t(* sf_crypto_signature_api_t::contextInit) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_signature_mode_t operation_mode, sf_crypto_signature_algorithm_init_params_t *const p_algorithm_specific_params, sf_crypto_key_t const *const p_key)

Perform Signature Module Context Initialization operation. This API initializes the signature module by setting operating mode, the message padding scheme and appropriate key for subsequent calls to signUpdate, verifyUpdate, signFinal and verifyFinal APIs. This API can be called only after this module has be opened using the open API. This API can be called after signFinal or verifyFinal API to initialize context for a new operation. This API sets up the internal context for sign/verify operation.

Implemented as

- SF_CRYPTO_SIGNATURE_ContextInit()

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	operation_mode	Selects Sign or Verify Operation enumeration.
[in]	p_algorithm_specific_params	Algorithm specific parameters.
[in]	p_key	Pointer to a private key for Sign operation OR Pointer to a public key for Verify operation.

Note

p_key should be WORD aligned.

◆ open

ssp_err_t(* sf_crypto_signature_api_t::open) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_signature_cfg_t const *const p_cfg)

Open SSP Crypto Signature framework. This function sets up a control block of the framework module based on the configuration parameters such as the key type, key size and domain parameters. The module allows users to have multiple instances with different control blocks, if required. This API will allocate memory internally according to the cryptography algorithm selected through key_type and key_size parameters.

Implemented as

- SF_CRYPTO_SIGNATURE_Open()

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	p_cfg	Pointer to sf_crypto_signature_cfg_t configuration structure. All elements of this structure must be set by user.

◆ signFinal

ssp_err_t(* sf_crypto_signature_api_t::signFinal) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message, sf_crypto_data_handle_t *const p_dest)

Perform Signature Module Signature-Final Operation. Call to this API generates signature and writes it to p_dest. p_message can be the pointer to last block of input message to be signed or can be passed as NULL if all of the input is passed through one or more signUpdate API call(s).

Implemented as

- SF_CRYPTO_SIGNATURE_SignFinal()

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	p_message	Pointer to data handle containing last block of data and its length. If there is no more data to be passed this param can be set to NULL.
[in,out]	p_dest	Pointer to data handle containing pointer to a buffer for storing signature. The data_length of this handle must be populated with the buffer length. Upon successful return this data_length will be updated with the number of bytes written to this buffer.

Note

p_message should be WORD aligned.
p_dest should be WORD aligned.
p_message can be set to NULL.
In case SF_CRYPTO_SIGNATURE_NO_PADDING is chosen as the padding scheme ensure p_message is a valid message digest in appropriate format.

◆ signUpdate

ssp_err_t(* sf_crypto_signature_api_t::signUpdate) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message)

Perform Signature Module Signature-Update operation. This API can be called multiple times to accumulate the message to be signed. This API can be used when the input message to be signed is not available all at once in a byte array.

Implemented as

- SF_CRYPTO_SIGNATURE_SignUpdate()

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	p_message	Pointer to input message to be signed.

Note

p_message should be WORD aligned.
In case SF_CRYPTO_SIGNATURE_NO_PADDING is chosen as the padding scheme ensure p_message is a valid message digest in appropriate format.

◆ verifyFinal

ssp_err_t(* sf_crypto_signature_api_t::verifyFinal) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_signature, sf_crypto_data_handle_t const *const p_message)

Perform Signature Module Signature-Verification-Final Operation. Call to this API performs signature verification operation. p_message can be the pointer to last block of message or can be passed as NULL if all of the message whose signature is to be verified is passed through one or more verifyUpdate API call(s).

Implemented as

- SF_CRYPTO_SIGNATURE_VerifyFinal()

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	p_signature	Pointer to Signature buffer to be verified.
[in]	p_message	Pointer to last block of message whose signature is being verified. If there is no more data to be passed this param can be set to NULL.

Note

p_message should be WORD aligned.
p_message can be set to NULL.

◆ verifyUpdate

ssp_err_t(* sf_crypto_signature_api_t::verifyUpdate) (sf_crypto_signature_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message)

Perform Signature Module Signature-Verification-Update operation. This API can be called multiple times to accumulate the message whose signature is to be verified. This API can be used when the input message to be verified against a signature is not available all at once in a byte array.

Implemented as

- SF_CRYPTO_SIGNATURE_VerifyUpdate()

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework control block structure.
[in]	p_message	Pointer to message whose signature is to be verified.

Note

p_message should be WORD aligned.

◆ versionGet

ssp_err_t(* sf_crypto_signature_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto Signature framework.

Implemented as

- SF_CRYPTO_SIGNATURE_VersionGet()

Parameters

[out]	p_version	Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_signature_api.h

sf_crypto_signature_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto Signature Framework Interface

#include <sf_crypto_signature_api.h>

Data Fields

sf_crypto_signature_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_signature_cfg_t *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_signature_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_signature_api.h

5.1.2.34 SSP Crypto TRNG Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Synergy Crypto TRNG Framework module. More...

Data Structures

struct	sf_crypto_trng_cfg_t
struct	sf_crypto_trng_api_t
struct	sf_crypto_trng_instance_t

Macros

#define	SF_CRYPTO_TRNG_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_crypto_trng_ctrl_t

Enumerations

enum sf_crypto_trng_state_t { SF_CRYPTO_TRNG_CLOSED = 0, SF_CRYPTO_TRNG_OPENED = 1 }

Detailed Description

Interface definition for Synergy Crypto TRNG Framework module.

Summary

This is the Interface of SF_CRYPTO_TRNG Framework module.

Crypto TRNG Framework Interface description: Crypto Framework

Macro Definition Documentation

◆ SF_CRYPTO_TRNG_API_VERSION_MAJOR

#define SF_CRYPTO_TRNG_API_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework

Typedef Documentation

◆ sf_crypto_trng_ctrl_t

typedef void sf_crypto_trng_ctrl_t
SSP Crypto TRNG framework control block. **Implemented as** ◦ sf_crypto_trng_ctrl_t

Enumeration Type Documentation

◆ sf_crypto_trng_state_t

enum sf_crypto_trng_state_t	
State codes for the SSP Crypto TRNG framework	
Enumerator	
SF_CRYPTO_TRNG_CLOSED	Crypto TRNG Framework Module is closed.
SF_CRYPTO_TRNG_OPENED	Crypto TRNG Framework Module is opened.

sf_crypto_trng_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto TRNG Framework Interface

#include <sf_crypto_trng_api.h>

Data Fields

sf_crypto_instance_t *	p_lower_lvl_common Pointer to a Crypto Framework common instance.
trng_instance_t *	p_lower_lvl_instance Pointer to Crypto TRNG HAL instance.
void *	p_extend Pointer to an optional configuration for HW specific settings.

Detailed Description

Configuration structure for the SSP Crypto TRNG framework

The documentation for this struct was generated from the following file:

- sf_crypto_trng_api.h

sf_crypto_trng_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto TRNG Framework Interface

#include <sf_crypto_trng_api.h>

Data Fields

ssp_err_t(*	open)(sf_crypto_trng_ctrl_t *const p_ctrl, sf_crypto_trng_cfg_t const *const p_cfg)

ssp_err_t(*	close)(sf_crypto_trng_ctrl_t *const p_ctrl)
ssp_err_t(*	randomNumberGenerate)(sf_crypto_trng_ctrl_t *const p_ctrl, sf_crypto_data_handle_t *const p_random_number_buff)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for the SSP Crypto framework

Field Documentation

◆ close

ssp_err_t(* sf_crypto_trng_api_t::close) (sf_crypto_trng_ctrl_t *const p_ctrl)

Close SSP Crypto TRNG framework. This API should be called once TRNG services are no longer needed.

Implemented as

- SF_CRYPTO_TRNG_Close()

Parameters

[in,out]	p_ctrl_api	Pointer to Crypto TRNG Framework control block structure.

◆ open

ssp_err_t(* sf_crypto_trng_api_t::open) (sf_crypto_trng_ctrl_t *const p_ctrl, sf_crypto_trng_cfg_t const *const p_cfg)

Open SSP Crypto TRNG framework for true random number generation.

Implemented as

- SF_CRYPTO_TRNG_Open()

Parameters

[in,out]	p_ctrl_api	Pointer to Crypto TRNG Framework control block structure.
[in]	p_cfg	Pointer to sf_crypto_trng_cfg_t configuration structure. All elements of this structure must be set by user.

◆ randomNumberGenerate

ssp_err_t(* sf_crypto_trng_api_t::randomNumberGenerate) (sf_crypto_trng_ctrl_t *const p_ctrl, sf_crypto_data_handle_t *const p_random_number_buff)

Generate a True Random Number of specified size

Implemented as

- SF_CRYPTO_TRNG_RandomNumberGenerate()

Parameters

[in]	p_ctrl_api	Pointer to Crypto TRNG Framework control block structure.
[in,out]	p_random_number_buff	Pointer to sf_crypto_data_handle_t structure storing pointer to buffer and its size where true random number will be returned.

Note

Size value specified under p_random_number_buff must be specified in Bytes.
Size value specified under p_random_number_buff must not be 0 Bytes. Its minimum value is 1.
Pointer to data buffer specified under p_random_number_buff must not be NULL.
Data buffer must be WORD aligned. The memory allocation to store the true random number is user's responsibility.

◆ versionGet

ssp_err_t(* sf_crypto_trng_api_t::versionGet) (ssp_version_t *const p_version)

Get version of SSP Crypto TRNG Framework Module.

Implemented as

- SF_CRYPTO_TRNG_VersionGet()

Parameters

[out]	p_version	Pointer to the memory to store the module version.

The documentation for this struct was generated from the following file:

- sf_crypto_trng_api.h

sf_crypto_trng_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SSP Crypto TRNG Framework Interface

#include <sf_crypto_trng_api.h>

Data Fields

sf_crypto_trng_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_crypto_trng_cfg_t *	p_cfg Pointer to the configuration structure for this instance.
sf_crypto_trng_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_crypto_trng_api.h

5.1.2.35 GUIX Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface definition for Adapting Microsoft GUIX for Synergy graphics drivers. More...

Data Structures

struct	sf_el_gx_callback_args_t
struct	sf_el_gx_cfg_t
struct	sf_el_gx_api_t

struct	sf_el_gx_instance_t

Macros

#define	SF_EL_GX_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_el_gx_ctrl_t

Enumerations

enum	sf_el_gx_state_t
enum	sf_el_gx_device_t { SF_EL_GX_DEVICE_NONE = 0, SF_EL_GX_DEVICE_DISPLAY = 1, SF_EL_GX_DEVICE_DRW = 2, SF_EL_GX_DEVICE_JPEG = 3 }
enum	sf_el_gx_event_t { SF_EL_GX_EVENT_ERROR = 1, SF_EL_GX_EVENT_DISPLAY_VSYNC = 2, SF_EL_GX_EVENT_UNDERFLOW = 3 }

Detailed Description

Interface definition for Adapting Microsoft GUIX for Synergy graphics drivers.

Summary

This is the Interface of SF_EL_GX Framework module which ties Synergy graphics device drivers to GUIX. The interface provides following driver adaptation for GUIX:

- DISPLAY driver for displaying image on LCD(e.g. GLCDC)
- Dave/2d driver for image rendering by 2DG engine
- JPEG driver for the image rendering by JPEG engine
- Software image rendering with no hardware acceleration

Implemented by: GUIX Synergy Port

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

GUIX Interface description: GUIX Port

Macro Definition Documentation

◆ SF_EL_GX_API_VERSION_MAJOR

#define SF_EL_GX_API_VERSION_MAJOR (2U)
The API version of GUIX integrated driver framework

Typedef Documentation

◆ sf_el_gx_ctrl_t

typedef void sf_el_gx_ctrl_t
GUIX adaptation framework control block. Allocate an instance specific control block to pass into the GUIX adaptation framework API calls. **Implemented as** ◦ sf_el_gx_instance_ctrl_t

Enumeration Type Documentation

◆ sf_el_gx_device_t

enum sf_el_gx_device_t	
Low level device code for the GUIX	
Enumerator	
SF_EL_GX_DEVICE_NONE	Non hardware.
SF_EL_GX_DEVICE_DISPLAY	Display device.
SF_EL_GX_DEVICE_DRW	2D Graphics Engine
SF_EL_GX_DEVICE_JPEG	JPEG Decoder.

◆ sf_el_gx_event_t

enum sf_el_gx_event_t	
Display event codes	
Enumerator	
SF_EL_GX_EVENT_ERROR	Low level driver error occurs.
SF_EL_GX_EVENT_DISPLAY_VSYNC	Display interface VSYNC.
SF_EL_GX_EVENT_UNDERFLOW	Display interface underflow.

◆ sf_el_gx_state_t

enum sf_el_gx_state_t
State codes for the SSP GUIX adaptation framework

sf_el_gx_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » GUIX Interface

#include <sf_el_gx_api.h>

Data Fields

sf_el_gx_device_t	device Device code.
sf_el_gx_event_t	event Event code of the low level hardware.
uint32_t	error Error code if SF_EL_GX_EVENT_ERROR.

Detailed Description

Callback arguments for the SSP GUIX adaptation framework

The documentation for this struct was generated from the following file:

- sf_el_gx_api.h

sf_el_gx_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » GUIX Interface

#include <sf_el_gx_api.h>

Data Fields

display_instance_t *	p_display_instance Pointer to a display instance.
display_runtime_cfg_t *	p_display_runtime_cfg Pointer to a runtime display configuration.
display_frame_layer_t	inherit_frame_layer Configured Inherit Screen Layer.
void *	p_canvas Pointer to a canvas(reserved)
void *	p_framebuffer_a Pointer to a frame buffer(A)
void *	p_framebuffer_b Pointer to a frame buffer(B)
void(*	p_callback)(sf_el_gx_callback_args_t *p_args) Pointer to callback function.
void *	p_context Pointer to a context.
void *	p_jpegbuffer Pointer to a JPEG work buffer.
uint32_t	jpegbuffer_size Size of a JPEG work buffer.
void *	p_sf_jpeg_decode_instance Pointer to a JPEG framework instance.

_Bool	dave2d_buffer_cache_enabled D/AVE 2D buffer cache enabled/disabled.

Detailed Description

Configuration structure for the SSP GUIX adaptation framework

---

The documentation for this struct was generated from the following file:

- sf_el_gx_api.h

sf_el_gx_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » GUIX Interface

#include <sf_el_gx_api.h>

Data Fields

ssp_err_t(*	open)(sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_el_gx_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)
UINT(*	setup)(GX_DISPLAY *p_display)
ssp_err_t(*	canvasInit)(sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)

Detailed Description

Shared Interface definition for the SSP GUIX adaptation framework

Field Documentation

◆ canvasInit

ssp_err_t(* sf_el_gx_api_t::canvasInit) (sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)

Canvas initialization. Set the memory address of the initial canvas.

Implemented as

- SF_EL_GX_CanvasInit()

Parameters

[in,out]	p_ctrl	Pointer to SF_EL_GX control block structure.
[in]	p_window_root	Pointer to GUIX root window context.

◆ close

ssp_err_t(* sf_el_gx_api_t::close) (sf_el_gx_ctrl_t *const p_ctrl)

Close SSP GUIX adaptation framework.

Implemented as

- SF_EL_GX_Close()

Parameters

[in,out]	p_ctrl	Pointer to SF_EL_GX control block structure.

◆ open

ssp_err_t(* sf_el_gx_api_t::open) (sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)

Open SSP GUIX adaptation framework.

Implemented as

- SF_EL_GX_Open()

Parameters

[in,out]	p_ctrl	Pointer to SF_EL_GX control block structure. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to SF_EL_GX configuration structure. All elements of this structure must be set by user.

◆ setup

UINT(* sf_el_gx_api_t::setup) (GX_DISPLAY *p_display)

Microsoft GUIX Driver setup entry.

Implemented as

- SF_EL_GX_Setup()

Parameters

[in]	p_display	Pointer to GUIX display driver setup function.

◆ versionGet

ssp_err_t(* sf_el_gx_api_t::versionGet) (ssp_version_t *p_version)

Get version.

Implemented as

- SF_EL_GX_VersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the version information.

The documentation for this struct was generated from the following file:

- sf_el_gx_api.h

sf_el_gx_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » GUIX Interface

#include <sf_el_gx_api.h>

Data Fields

sf_el_gx_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_el_gx_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_el_gx_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_el_gx_api.h

5.1.2.36 External IRQ Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated External IRQ Framework Interface. More...

Data Structures

struct	sf_external_irq_cfg_t
struct	sf_external_irq_api_t
struct	sf_external_irq_instance_t

Typedefs

typedef void	sf_external_irq_ctrl_t

Enumerations

enum	sf_external_irq_event_t { SF_EXTERNAL_IRQ_EVENT_NONE, SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT }

Detailed Description

RTOS-integrated External IRQ Framework Interface.

Summary

This module is a ThreadX-aware external IRQ Framework Interface for external inputs such as switches or other binary signals. The Interface is implemented by External IRQ Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

External IRQ Framework Interface description: External IRQ Framework

Typedef Documentation

◆ sf_external_irq_ctrl_t

typedef void sf_external_irq_ctrl_t

External interrupt control block. Allocate an instance specific control block to pass into the external interrupt framework API calls.

Implemented as

- sf_external_irq_instance_ctrl_t

Enumeration Type Documentation

◆ sf_external_irq_event_t

enum sf_external_irq_event_t	
Options for what should happen when the external interrupt expires.	
Enumerator	
SF_EXTERNAL_IRQ_EVENT_NONE	Nothing happens during expiration. Can be used for data transfer.
SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT	Posts to internal semaphore. Select this if using SF_EXTERNAL_IRQ_Wait.

sf_external_irq_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » External IRQ Framework Interface

#include <sf_external_irq_api.h>

Data Fields

external_irq_instance_t const *	p_lower_lvl_irq
sf_external_irq_event_t	event Select what happens when the external IRQ is triggered.

Detailed Description

Configuration for RTOS integrated external interrupt driver

Field Documentation

◆ p_lower_lvl_irq

external_irq_instance_t const* sf_external_irq_cfg_t::p_lower_lvl_irq
All info needed to work with lower layer

The documentation for this struct was generated from the following file:

- sf_external_irq_api.h

sf_external_irq_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » External IRQ Framework Interface

#include <sf_external_irq_api.h>

Data Fields

ssp_err_t(*	open)(sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)
ssp_err_t(*	wait)(sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	close)(sf_external_irq_ctrl_t *const p_ctrl)

Detailed Description

External IRQ framework API structure. External IRQ implementations use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_external_irq_api_t::close) (sf_external_irq_ctrl_t *const p_ctrl)

Close channel at HAL layer and release the RTOS services.

Implemented as

- SF_EXTERNAL_IRQ_Close()

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for this external interrupt.

◆ open

ssp_err_t(* sf_external_irq_api_t::open) (sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)

Create the semaphore, then handle driver initialization at the HAL layer.

Implemented as

- SF_EXTERNAL_IRQ_Open()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. The device control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ versionGet

ssp_err_t(* sf_external_irq_api_t::versionGet) (ssp_version_t *const p_version)

Get version and store it in provided pointer p_version.

Implemented as

- SF_EXTERNAL_IRQ_VersionGet()

Parameters

[out]	p_version	Code and API version used stored here.

◆ wait

ssp_err_t(* sf_external_irq_api_t::wait) (sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)

Wait for the next external interrupt expiration, then return.

Precondition

Call SF_EXTERNAL_IRQ_Open to configure the external IRQ before using this function. During SF_EXTERNAL_IRQ_Open, set sf_external_irq_cfg_t::sf_external_irq_event_t to SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT.

Implemented as

- SF_EXTERNAL_IRQ_Wait()

Parameters

[in]	p_ctrl	Handle set in SF_EXTERNAL_IRQ_Open.
[in]	timeout	ThreadX timeout. Select TX_NO_WAIT, a value in system clock counts between 1 and 0xFFFFFFFF, or TX_WAIT_FOREVER.

The documentation for this struct was generated from the following file:

- sf_external_irq_api.h

sf_external_irq_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » External IRQ Framework Interface

#include <sf_external_irq_api.h>

Data Fields

sf_external_irq_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_external_irq_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.

sf_external_irq_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_external_irq_api.h

5.1.2.37 I2C Framework

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated I2C Framework Interface. More...

Data Structures

struct	sf_i2c_bus_t
struct	sf_i2c_cfg_t
struct	sf_i2c_api_t
struct	sf_i2c_instance_t

Macros

#define	SF_I2C_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_i2c_ctrl_t

Enumerations

enum	sf_i2c_dev_state_t { SF_I2C_DEV_STATE_CLOSED = 0, SF_I2C_DEV_STATE_OPENED }

Detailed Description

RTOS-integrated I2C Framework Interface.

Summary

This is a ThreadX-aware I2C interface. It can be implemented by several hardware peripherals at the HAL layer through the I2C interface I2C Interface.

The connection to the HAL layer is established by passing in a driver structure in SF_I2C_Open().

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SPI Framework Interface description: I2C Framework

Macro Definition Documentation

◆ SF_I2C_API_VERSION_MAJOR

#define SF_I2C_API_VERSION_MAJOR (2U)
Includes driver interface.

Typedef Documentation

◆ sf_i2c_ctrl_t

typedef void sf_i2c_ctrl_t
I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. **Implemented as** ◦ sf_i2c_instance_ctrl_t

Enumeration Type Documentation

◆ sf_i2c_dev_state_t

enum sf_i2c_dev_state_t	
SF I2C device state	
Enumerator	
SF_I2C_DEV_STATE_CLOSED	I2C device is closed.
SF_I2C_DEV_STATE_OPENED	I2C device is opened.

sf_i2c_bus_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » I2C Framework

#include <sf_i2c_api.h>

Data Fields

uint8_t	channel Channel.
TX_MUTEX *	p_lock_mutex Lock mutex handle for this channel.
TX_MUTEX	device_count_mutex Device count mutex handle for this device.
TX_EVENT_FLAGS_GROUP *	p_sync_eventflag Pointer to the event flag object for I2C data transfer.
sf_i2c_ctrl_t **	pp_curr_ctrl Current device using the bus (by switching the address)
uint8_t *	p_bus_name User-supplied name to identify the bus. Useful for debugging.
i2c_api_master_t const *	p_lower_lvl_api Pointer to I2C HAL interface to be used in the framework.
uint8_t	device_count Number of devices on the bus; initialize to 0.
sf_i2c_ctrl_t **	pp_curr_bus_ctrl Device configured on the bus (low level configuration)

Detailed Description

Data structure defining a I2C bus.

The documentation for this struct was generated from the following file:

- sf_i2c_api.h

sf_i2c_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » I2C Framework

#include <sf_i2c_api.h>

Data Fields

sf_i2c_bus_t *	p_bus Bus used by the device.
i2c_cfg_t const *	p_lower_lvl_cfg Pointer to I2C HAL configuration.

Detailed Description

Configuration for Framework I2C driver

The documentation for this struct was generated from the following file:

- sf_i2c_api.h

sf_i2c_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » I2C Framework

#include <sf_i2c_api.h>

Data Fields

ssp_err_t(*	open)(sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)

	Open a designated I2C device on a bus. More...
ssp_err_t(*	read)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout) Receive data from I2C device. More...
ssp_err_t(*	write)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout) Transmit data to I2C device. More...
ssp_err_t(*	reset)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout) Abort any in-progress transfer and force the I2C peripheral into a ready state. More...
ssp_err_t(*	close)(sf_i2c_ctrl_t *const p_ctrl) Disable the I2C device designated by the control handle. Close the RTOS services used by the bus if no devices are connected to the bus. More...
ssp_err_t(*	lock)(sf_i2c_ctrl_t *const p_ctrl) Lock the bus for a device. Locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances. More...
ssp_err_t(*	unlock)(sf_i2c_ctrl_t *const p_ctrl) Unlock the bus from a particular device and make it available for other devices. This allows other devices to use bus for reads and writes on the bus. More...
ssp_err_t(*	version)(ssp_version_t *const p_version) Get I2C framework version. More...
ssp_err_t(*	lockWait)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout) Lock the I2C bus for a device. Locking reserves exclusive access to the I2C driver. Here a wait option is provided to the user. This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances. More...

Detailed Description

Shared Interface definition for I2C Framework

Field Documentation

◆ close

ssp_err_t(* sf_i2c_api_t::close) (sf_i2c_ctrl_t *const p_ctrl)

Disable the I2C device designated by the control handle. Close the RTOS services used by the bus if no devices are connected to the bus.

Implemented as

- SF_I2C_Close()

Parameters

[in]	p_ctrl	Control handle for I2C framework driver context for a device

◆ lock

ssp_err_t(* sf_i2c_api_t::lock) (sf_i2c_ctrl_t *const p_ctrl)

Lock the bus for a device. Locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances.

Implemented as

- SF_I2C_Lock()

Parameters

[in]	p_ctrl	Control handle for I2C framework driver context for a device

◆ lockWait

ssp_err_t(* sf_i2c_api_t::lockWait) (sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)

Lock the I2C bus for a device. Locking reserves exclusive access to the I2C driver. Here a wait option is provided to the user. This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances.

Implemented as

- SF_I2C_LockWait()

Parameters

[in]	p_ctrl	Control handle for I2C framework driver context for a device
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ open

ssp_err_t(* sf_i2c_api_t::open) (sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)

Open a designated I2C device on a bus.

Implemented as

- SF_I2C_Open()

Parameters

[out]	p_ctrl	Control handle for I2C framework driver context for a device (Value returns from this function). This value must be cleared by user.
[in]	p_cfg	I2C configuration includes I2C bus and low level configuration

◆ read

ssp_err_t(* sf_i2c_api_t::read) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)

Receive data from I2C device.

Implemented as

- SF_I2C_Read()

Parameters

[in]	p_ctrl	Pointer to previously opened I2C SF control structure.
[in]	p_dest	Pointer to location to store read data.
[in]	bytes	Number of bytes to read.
[in]	restart	Indicates whether the restart condition should be issued after reading.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ reset

ssp_err_t(* sf_i2c_api_t::reset) (sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)

Abort any in-progress transfer and force the I2C peripheral into a ready state.

Implemented as

- SF_I2C_Reset()

This function safely terminates any in-progress I2C transfer with the device. If a transfer is aborted, the user is be notified via callback with an abort event. Since the callback is optional, this function also returns a specific error code in this situation.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for I2C driver.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ unlock

ssp_err_t(* sf_i2c_api_t::unlock) (sf_i2c_ctrl_t *const p_ctrl)

Unlock the bus from a particular device and make it available for other devices. This allows other devices to use bus for reads and writes on the bus.

Implemented as

- SF_I2C_Unlock()

Parameters

[in]	p_ctrl	Control handle for I2C framework driver context for a device

◆ version

ssp_err_t(* sf_i2c_api_t::version) (ssp_version_t *const p_version)

Get I2C framework version.

Implemented as

- SF_I2C_VersionGet()

Parameters

[in]	p_ctrl	Handle for I2C framework control block for a device

◆ write

ssp_err_t(* sf_i2c_api_t::write) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)

Transmit data to I2C device.

Implemented as

- SF_I2C_Write()

Parameters

[in]	p_ctrl	Pointer to previously opened I2C control structure.
[in]	p_src	Pointer to location to get write data.
[in]	bytes	Number of bytes to write.
[in]	restart	Indicates whether the restart condition should be issued after writing.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

The documentation for this struct was generated from the following file:

- sf_i2c_api.h

sf_i2c_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » I2C Framework

#include <sf_i2c_api.h>

Data Fields

sf_i2c_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_i2c_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_i2c_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_i2c_api.h

5.1.2.38 JPEG Decode Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated JPEG Decode Framework Interface. More...

Data Structures

struct	sf_jpeg_decode_cfg_t
struct	sf_jpeg_decode_api_t
struct	sf_jpeg_decode_instance_t

Macros

#define	SF_JPEG_DECODE_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_jpeg_decode_ctrl_t

Detailed Description

RTOS-integrated JPEG Decode Framework Interface.

Summary

This is a ThreadX aware generic JPEG decoding framework for run-time JPEG decode applications. It can be implemented by either hardware or software. For Synergy parts, the interface is implemented by the on-chip JPEG decoding engine. The connection to the HAL layer is established by passing in a driver structure in SF_JPEG_Decode_Open.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Framework JPEG Decode Interface description: JPEG Decode Framework

Macro Definition Documentation

◆ SF_JPEG_DECODE_API_VERSION_MAJOR

#define SF_JPEG_DECODE_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_jpeg_decode_ctrl_t

typedef void sf_jpeg_decode_ctrl_t
JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. **Implemented as** ◦ sf_jpeg_decode_instance_ctrl_t

sf_jpeg_decode_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » JPEG Decode Framework Interface

#include <sf_jpeg_decode_api.h>

Data Fields

jpeg_decode_instance_t const *	p_lower_lvl_jpeg_decode

Detailed Description

Configuration for RTOS integrated JPEG driver

Field Documentation

◆ p_lower_lvl_jpeg_decode

jpeg_decode_instance_t const* sf_jpeg_decode_cfg_t::p_lower_lvl_jpeg_decode

Pointer to a driver structure that implements this interface. Pre-configured driver structures are located in r_jpeg_decode.c and extern'ed in r_jpeg_decode.h.

The documentation for this struct was generated from the following file:

- sf_jpeg_decode_api.h

sf_jpeg_decode_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » JPEG Decode Framework Interface

#include <sf_jpeg_decode_api.h>

Data Fields

ssp_err_t(*	open)(sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)
ssp_err_t(*	inputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)
ssp_err_t(*	outputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer,

	uint32_t buffer_size)
ssp_err_t(*	linesDecodedGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
ssp_err_t(*	horizontalStrideSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
ssp_err_t(*	imageSubsampleSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
ssp_err_t(*	wait)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout)
ssp_err_t(*	statusGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
ssp_err_t(*	imageSizeGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
ssp_err_t(*	pixelFormatGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)
ssp_err_t(*	close)(sf_jpeg_decode_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

JPEG Decode API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_jpeg_decode_api_t::close) (sf_jpeg_decode_ctrl_t *const p_ctrl)

Closes JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded.

Parameters

[in]	p_ctrl	Pointer to the control block set in SF_JPEG_Decode_Open().

◆ horizontalStrideSet

ssp_err_t(* sf_jpeg_decode_api_t::horizontalStrideSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

Configure the horizontal stride value.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	horizontal_stride	Set the horizontal stride value, in pixels

◆ imageSizeGet

ssp_err_t(* sf_jpeg_decode_api_t::imageSizeGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Obtain the size of the image. This function is only useful for decoding a JPEG image.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	p_horizontal_size	Width of the image, in pixels
[out]	p_vertical_size	Height of the image, in pixels

◆ imageSubsampleSet

ssp_err_t(* sf_jpeg_decode_api_t::imageSubsampleSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[in]	horizontal_subsample	Set the horizontal subsample value
[in]	vertical_subsample	Set the vertical subsample value

◆ inputBufferSet

ssp_err_t(* sf_jpeg_decode_api_t::inputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)

Feed data into JPEG codec module.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[in]	p_buffer	Buffer contains data to be processed by the JPEG codec module. The buffer starting address must be 8-byte aligned
[in]	buffer_size	Size of the data buffer, must be multiple of 8 bytes

◆ linesDecodedGet

ssp_err_t(* sf_jpeg_decode_api_t::linesDecodedGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)

Obtain number of lines JPEG codec decoded.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	p_lines	Number of lines decoded into the output buffer.

◆ open

ssp_err_t(* sf_jpeg_decode_api_t::open) (sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)

Acquire mutex, then handle driver initialization at the HAL layer. This function releases mutex before it returns to the caller.

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. Elements initialized here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ outputBufferSet

ssp_err_t(* sf_jpeg_decode_api_t::outputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Read processed data from JPEG codec module.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[in]	p_buffer	User-supplied buffer space to hold output from JPEG codec module. The buffer starting address must be 8-byte aligned
[in]	buffer_size	Size of the output data buffer

◆ pixelFormatGet

ssp_err_t(* sf_jpeg_decode_api_t::pixelFormatGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

Obtain the pixel format of the image. This function is only useful for decoding a JPEG image.

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	p_color_space	Color space of the image

◆ statusGet

ssp_err_t(* sf_jpeg_decode_api_t::statusGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)

Obtain JPEG codec status

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	p_status	Status of current JPEG codec module

◆ versionGet

ssp_err_t(* sf_jpeg_decode_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

◆ wait

ssp_err_t(* sf_jpeg_decode_api_t::wait) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout)

Wait for current JPEG codec operation to finish

Parameters

[in]	p_ctrl	Pointer to the control block initialized in SF_JPEG_Decode_Open().
[out]	p_status	Status of current JPEG codec module
[out]	timeout	Amount of time (in ThreadX ticks) to wait

The documentation for this struct was generated from the following file:

- sf_jpeg_decode_api.h

sf_jpeg_decode_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » JPEG Decode Framework Interface

#include <sf_jpeg_decode_api.h>

Data Fields

sf_jpeg_decode_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_jpeg_decode_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_jpeg_decode_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_jpeg_decode_api.h

5.1.2.39 Memory interface

Renesas Synergy Software Package Reference » Framework Interfaces

Interface for Memory API. More...

Data Structures

struct	sf_memory_region_info_t

struct	sf_memory_info_t
struct	sf_memory_cfg_t
struct	sf_memory_api_t
struct	sf_memory_instance_t

Macros

#define	SF_MEMORY_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_memory_ctrl_t

Detailed Description

Interface for Memory API.

Summary

The Memory interface supports the following features.

- Read from any memory
- Erase memory
- Write to Memory
- Retrieve device specific information

Implemented by: Memory framework

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Memory Interface description: Memory Framework on sf_memory_qspi_nor

Macro Definition Documentation

◆ SF_MEMORY_API_VERSION_MAJOR

#define SF_MEMORY_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ sf_memory_ctrl_t

typedef void sf_memory_ctrl_t
SF Memory API framework control block. Allocate an instance specific control block to pass into the framework API calls. **Implemented as** ◦ sf_memory_qspi_nor_instance_ctrl_t

sf_memory_region_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Memory interface

#include <sf_memory_api.h>

Data Fields

uint32_t	memory_start_address Start address of this memory region.
uint32_t	memory_end_address End address of this memory region.
uint32_t	minimum_erase_size Specify the minimum erase size of the memory type.
uint32_t	minimum_write_size Specify the minimum write size of the memory type.

Detailed Description

Representation of a memory layout

The documentation for this struct was generated from the following file:

- sf_memory_api.h

sf_memory_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Memory interface

#include <sf_memory_api.h>

Data Fields

uint32_t	number_of_regions Number of memory regions supported by this memory type.
sf_memory_region_info_t *	p_regions_info Memory region specific information.

Detailed Description

Memory information supported by the instance

The documentation for this struct was generated from the following file:

- sf_memory_api.h

sf_memory_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Memory interface

#include <sf_memory_api.h>

Data Fields

void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- sf_memory_api.h

sf_memory_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Memory interface

#include <sf_memory_api.h>

Data Fields

ssp_err_t(*	open)(sf_memory_ctrl_t *const p_api_ctrl, sf_memory_cfg_t const *const p_cfg)
ssp_err_t(*	read)(sf_memory_ctrl_t *const p_api_ctrl, uint8_t *const p_dest_address, uint32_t const memory_address, uint32_t const num_bytes)
ssp_err_t(*	write)(sf_memory_ctrl_t *const p_api_ctrl, uint8_t *const p_src_address, uint32_t const memory_address, uint32_t const num_bytes)
ssp_err_t(*	flush)(sf_memory_ctrl_t *const p_api_ctrl)
ssp_err_t(*	erase)(sf_memory_ctrl_t *const p_api_ctrl, uint32_t const memory_address, uint32_t const num_bytes)
ssp_err_t(*	infoGet)(sf_memory_ctrl_t *const p_api_ctrl, sf_memory_info_t *const p_info)
ssp_err_t(*	close)(sf_memory_ctrl_t *const p_api_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

SF_MEMORY functions implemented at the Framework layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* sf_memory_api_t::close) (sf_memory_ctrl_t *const p_api_ctrl)

Closes the module.

Implemented as

- SF_MEMORY_QSPI_NOR_Close

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.

◆ erase

ssp_err_t(* sf_memory_api_t::erase) (sf_memory_ctrl_t *const p_api_ctrl, uint32_t const memory_address, uint32_t const num_bytes)

Erases the specified number of bytes from the memory device.

Implemented as

- SF_MEMORY_QSPI_NOR_Erase

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.
[in]	memory_address	Address to start the erase process at.
[in]	num_bytes	Number of bytes of data to erase.

◆ flush

ssp_err_t(* sf_memory_api_t::flush) (sf_memory_ctrl_t *const p_api_ctrl)

Performs any buffered pending writes. This function MUST be called by the application after the final write is called to complete any pending writes.

Implemented as

- SF_MEMORY_QSPI_NOR_Flush

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.

◆ infoGet

ssp_err_t(* sf_memory_api_t::infoGet) (sf_memory_ctrl_t *const p_api_ctrl, sf_memory_info_t *const p_info)

Returns information about the memory implementation.

Implemented as

- SF_MEMORY_QSPI_NOR_InfoGet

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.
[out]	p_info	Pointer to information structure. All elements of this structure will be set by the function.

◆ open

ssp_err_t(* sf_memory_api_t::open) (sf_memory_ctrl_t *const p_api_ctrl, sf_memory_cfg_t const *const p_cfg)

Initialize Memory framework.

Implemented as

- SF_MEMORY_QSPI_NOR_Open

Parameters

[in]	p_api_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_config	Pointer to configuration structure. All elements of this structure must be set by user.

◆ read

ssp_err_t(* sf_memory_api_t::read) (sf_memory_ctrl_t *const p_api_ctrl, uint8_t *const p_dest_address, uint32_t const memory_address, uint32_t const num_bytes)

Reads data from the specified memory device address to the location specified by the caller.

Implemented as

- SF_MEMORY_QSPI_NOR_Read

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.
[in]	p_dest_address	Destination to read the data into.
[in]	memory_address	Address to read the data from.
[in]	num_bytes	Number of bytes of data to read.

◆ versionGet

ssp_err_t(* sf_memory_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Implemented as

- SF_MEMORY_QSPI_NOR_VersionGet

Parameters

[out]	p_version	Code and API version used.

◆ write

ssp_err_t(* sf_memory_api_t::write) (sf_memory_ctrl_t *const p_api_ctrl, uint8_t *const p_src_address, uint32_t const memory_address, uint32_t const num_bytes)

Writes the specified number of data bytes to the specified device memory address.

Implemented as

- SF_MEMORY_QSPI_NOR_Write

Parameters

[in]	p_api_ctrl	Control block set in sf_memory_api_t::open call.
[in]	p_src_address	Address to read the data to be written.
[in]	memory_address	Address to write the data to.
[in]	num_bytes	Number of bytes of data to write.

The documentation for this struct was generated from the following file:

- sf_memory_api.h

sf_memory_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Memory interface

#include <sf_memory_api.h>

Data Fields

sf_memory_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_memory_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_memory_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_memory_api.h

5.1.2.40 Messaging Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Messaging Framework Interface. More...

Data Structures

struct	sf_message_header_t
struct	sf_message_instance_range_t
struct	sf_message_subscriber_t
struct	sf_message_subscriber_list_t
struct	sf_message_callback_args_t
struct	sf_message_post_err_t
struct	sf_message_buffer_ctrl_t Buffer control block structure. More...
struct	sf_message_cfg_t
struct	sf_message_acquire_cfg_t
struct	sf_message_post_cfg_t
struct	sf_message_api_t
struct	sf_message_instance_t

Macros

#define	SF_MESSAGE_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_message_ctrl_t

Enumerations

enum	sf_message_state_t { SF_MESSAGE_STATE_CLOSED = 0, SF_MESSAGE_STATE_OPENED }
enum	sf_message_callback_event_t { SF_MESSAGE_CALLBACK_EVENT_ACK = 0, SF_MESSAGE_CALLBACK_EVENT_NAK }
enum	sf_message_priority_t { SF_MESSAGE_PRIORITY_NORMAL = 0, SF_MESSAGE_PRIORITY_HIGH }
enum	sf_message_release_option_t { SF_MESSAGE_RELEASE_OPTION_NONE = 0, SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE = 1, SF_MESSAGE_RELEASE_OPTION_ACK = 2, SF_MESSAGE_RELEASE_OPTION_NAK = 4 }

Detailed Description

RTOS-integrated Messaging Framework Interface.

Summary

This module is a ThreadX-aware Messaging Framework. This Interface is implemented by Messaging Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Messaging Interface description: Messaging Framework

Macro Definition Documentation

◆ SF_MESSAGE_API_VERSION_MAJOR

#define SF_MESSAGE_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_message_ctrl_t

typedef void sf_message_ctrl_t
Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. **Implemented as** ◦ sf_message_instance_ctrl_t

Enumeration Type Documentation

◆ sf_message_callback_event_t

enum sf_message_callback_event_t	
Messaging callback response	
Enumerator	
SF_MESSAGE_CALLBACK_EVENT_ACK	ACK response.
SF_MESSAGE_CALLBACK_EVENT_NAK	NAK response.

◆ sf_message_priority_t

enum sf_message_priority_t	
Messaging framework state	
Enumerator	
SF_MESSAGE_PRIORITY_NORMAL	Gives a message to be sent normal priority.
SF_MESSAGE_PRIORITY_HIGH	Gives a message to be sent higher priority than previous ones.

◆ sf_message_release_option_t

enum sf_message_release_option_t	
Messaging option	
Enumerator	
SF_MESSAGE_RELEASE_OPTION_NONE	No option.
SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE	Buffer forced release option.
SF_MESSAGE_RELEASE_OPTION_ACK	ACK response (note if both ACK and NAK are set at same time, NAK.
SF_MESSAGE_RELEASE_OPTION_NAK	NAK response.

◆ sf_message_state_t

enum sf_message_state_t	
Messaging framework state	
Enumerator	
SF_MESSAGE_STATE_CLOSED	Messaging framework is closed.
SF_MESSAGE_STATE_OPENED	Messaging framework is opened.

sf_message_header_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Detailed Description

Message header definition

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_instance_range_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

uint8_t	start Start of the event class instance range.
uint8_t	end End of the event class instance range.

Detailed Description

Subscriber lists definitions

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_subscriber_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

TX_QUEUE *	p_queue Pointer to the message queue for subscriber thread.
sf_message_instance_range_t	instance_range Range of the event class instance to receive message.

Detailed Description

Message subscriber

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_subscriber_list_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

sf_message_event_class_t	event_class Event class code.
uint16_t	number_of_nodes Number of nodes in the subscriber group.
sf_message_subscriber_t **	pp_subscriber_group Subscriber group for the event class.

Detailed Description

Message subscriber list

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework

Interface

#include <sf_message_api.h>

Data Fields

sf_message_callback_event_t	event Event code.
void const *	p_context Context provided to user during callback.

Detailed Description

Message framework callback parameters

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_post_err_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

TX_QUEUE *	p_queue Queue.

Detailed Description

Post error information structure

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_buffer_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

Buffer control block structure. More...

#include <sf_message_api.h>

Data Structures

struct	st_buffer_ctrl_flag

Data Fields

void(*	p_callback)(sf_message_callback_args_t *) Optional user callback function.
void const *	p_context Context provided to user during callback.

Detailed Description

Buffer control block structure.

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_buffer_ctrl_t::st_buffer_ctrl_flag Struct Reference

#include <sf_message_api.h>

Data Fields

uint32_t	semaphore: 16 Counting semaphore to prevent a buffer from being released.

uint32_t	buffer_keep: 1 Buffer keep request.
uint32_t	nak_response: 1 NAK (ORed logic for multiple subscribers)
uint32_t	reserved: 5 Reserved bits.
uint32_t	in_use: 1 Buffer in-use.

Detailed Description

Flags

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

void *	p_work_memory_start Start address of the memory area.
uint32_t	work_memory_size_bytes Size of working memory area in bytes.
uint32_t	buffer_size

	Bytes of the message block.
sf_message_subscriber_list_t **	pp_subscriber_lists Pointer array to the subscriber lists.
uint8_t *	p_block_pool_name Pointer to the block pool name.

Detailed Description

Messaging framework configuration structure definition

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_acquire_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

bool	buffer_keep Buffer keep option.

Detailed Description

Messaging framework Post API function configuration structure definition

---

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_post_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

sf_message_priority_t	priority Message priority.
void(*	p_callback)(sf_message_callback_args_t *) User callback function.
void const *	p_context Context provided to user during callback.

Detailed Description

Messaging framework Acquire API function configuration structure definition

---

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

ssp_err_t(*	open)(sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_message_ctrl_t *const p_ctrl)

ssp_err_t(*	bufferAcquire)(sf_message_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option)
ssp_err_t(*	bufferRelease)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option)
ssp_err_t(*	post)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
ssp_err_t(*	pend)(sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Messaging Framework API structure. Implementations will use the following API.

Field Documentation

◆ bufferAcquire

ssp_err_t(* sf_message_api_t::bufferAcquire) (sf_message_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option)

Acquire buffer for message passing from the block.

Implemented as

- SF_MESSAGE_BufferAcquire()

Parameters

[in]	p_ctrl	Pointer to the messaging control block
[in,out]	pp_buffer	Pointer to the pointer to the allocated buffer memory
[in]	p_acquire_cfg	Pointer to the buffer acquisition configuration
[in]	wait_option	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

◆ bufferRelease

ssp_err_t(* sf_message_api_t::bufferRelease) (sf_message_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option)

Release buffer obtained from SF_MESSAGE_BufferAcquire().

Implemented as

- SF_MESSAGE_BufferRelease()

Parameters

[in]	p_ctrl	Pointer to the messaging control block
[in]	p_buffer	Pointer to the buffer allocated by SF_MESSAGE_BufferAcquire()
[in]	option	Buffer release option (SF_MESSAGE_RELEASE_OPTION_NONE, SF_MESSAGE_RELEASE_OPTION_ACK, SF_MESSAGE_RELEASE_OPTION_NAK, SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE)

◆ close

ssp_err_t(* sf_message_api_t::close) (sf_message_ctrl_t *const p_ctrl)

Finalize message framework.

Implemented as

- SF_MESSAGE_Close()

Parameters

[in,out]	p_ctrl	Pointer to the messaging control block

◆ open

ssp_err_t(* sf_message_api_t::open) (sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)

Initialize message framework. Initiate the messaging framework control block, configure the work memory corresponding to the configuration parameters.

Implemented as

- SF_MESSAGE_Open()

Parameters

[in,out]	p_ctrl	Pointer to the messaging control block
[in]	p_cfg	Pointer to configuration structure

◆ pend

ssp_err_t(* sf_message_api_t::pend) (sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)

Pend message.

Implemented as

- SF_MESSAGE_Pend()

Parameters

[in]	p_ctrl	Pointer to the messaging control block
[in]	p_queue	Pointer to a threadX message queue object
[in,out]	pp_buffer	Pointer to the pointer to the buffer where message is stored.
[in]	wait_option	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

◆ post

ssp_err_t(* sf_message_api_t::post) (sf_message_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)

Post message to the subscribers.

Implemented as

- SF_MESSAGE_Post()

Parameters

[in]	p_ctrl	Pointer to the messaging control block
[in]	p_buffer	Pointer to the buffer allocated by SF_MESSAGE_BufferAcquire()
[in]	p_post_cfg	Pointer to the message post configuration
[in]	wait_option	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

◆ versionGet

ssp_err_t(* sf_message_api_t::versionGet) (ssp_version_t *const p_version)

Get the version of the messaging framework.

Implemented as

- SF_MESSAGE_VersionGet()

Parameters

[in]	p_version	Pointer to the memory where to store the version number

---

The documentation for this struct was generated from the following file:

- sf_message_api.h

sf_message_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Messaging Framework Interface

#include <sf_message_api.h>

Data Fields

sf_message_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_message_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_message_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_message_api.h

5.1.2.41 Power Profiles V2 Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

Power Profiles Framework Interface. More...

Data Structures

struct	sf_power_profiles_v2_callback_args_t
struct	sf_power_profiles_v2_ctrl_t
struct	sf_power_profiles_v2_cfg_t
struct	sf_power_profiles_v2_run_cfg_t

struct	sf_power_profiles_v2_low_power_cfg_t
struct	sf_power_profiles_v2_api_t
struct	sf_power_profiles_v2_instance_t

Macros

#define	SF_POWER_PROFILES_V2_API_VERSION_MAJOR (3U)

Enumerations

enum	sf_power_profiles_v2_event_t { SF_POWER_PROFILES_V2_EVENT_PRE_LOW_POWER, SF_POWER_PROFILES_V2_EVENT_POST_LOW_POWER }

Detailed Description

Power Profiles Framework Interface.

Summary

This framework allows an application to apply power profiles at runtime. There are 2 types of profiles: Run and Low Power. Applying a Run profile will change things like the system clock and IOPORT settings. The MCU will continue to run during this process and will not be put into a low power mode. Applying a Low Power profile will put the MCU into a low power mode. Which low power mode is used is specified by the LPMv2 instance used. IOPORT settings can also be specified which will be applied before entering the low power mode and after waking up.

The Deep Software Standby low power mode, available on some MCUs, will reset the MCU when the woken up. In this case the callback will not be called and the IOPORT configuration will not be applied after waking up.

This framework can be used with, or without, an RTOS.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Macro Definition Documentation

◆ SF_POWER_PROFILES_V2_API_VERSION_MAJOR

#define SF_POWER_PROFILES_V2_API_VERSION_MAJOR (3U)
Version of the API defined in this file

Enumeration Type Documentation

◆ sf_power_profiles_v2_event_t

enum sf_power_profiles_v2_event_t	
Options for the callback events.	
Enumerator	
SF_POWER_PROFILES_V2_EVENT_PRE_LOW_POWER	Callback just before entering low power mode.
SF_POWER_PROFILES_V2_EVENT_POST_LOW_POWER	Callback just after exiting the low power mode.

sf_power_profiles_v2_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

sf_power_profiles_v2_event_t	event Power profiles callback event.
void *	p_context Placeholder for user data.

Detailed Description

Power profiles callback arguments definitions

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

uint32_t	open Used by driver to check if pointer to control block is valid.

Detailed Description

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

---

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

void const *	p_extend

Detailed Description

Initialization configuration

Field Documentation

◆ p_extend

void const* sf_power_profiles_v2_cfg_t::p_extend
Pointer to additional settings (not currently in use)

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_run_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

ioport_cfg_t const *	p_ioport_pin_tbl
cgc_clocks_cfg_t const *	p_clock_cfg
void const *	p_extend

Detailed Description

Run profile configuration

Field Documentation

◆ p_clock_cfg

cgc_clocks_cfg_t const* sf_power_profiles_v2_run_cfg_t::p_clock_cfg
Pointer to a CGC configuration

◆ p_extend

void const* sf_power_profiles_v2_run_cfg_t::p_extend
Pointer to additional settings

◆ p_ioport_pin_tbl

ioport_cfg_t const* sf_power_profiles_v2_run_cfg_t::p_ioport_pin_tbl
Pointer to IOPORT settings

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_low_power_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

ioport_cfg_t const *	p_ioport_pin_tbl_exit
ioport_cfg_t const *	p_ioport_pin_tbl_enter
lpmv2_instance_t const *	p_lower_lvl_lpm
void(*	p_callback)(sf_power_profiles_v2_callback_args_t *p_args)
void *	p_context
void const *	p_extend

Detailed Description

Low Power profile configuration

Field Documentation

◆ p_callback

void(* sf_power_profiles_v2_low_power_cfg_t::p_callback) (sf_power_profiles_v2_callback_args_t *p_args)
Callback function

◆ p_context

void* sf_power_profiles_v2_low_power_cfg_t::p_context
Placeholder for user data

◆ p_extend

void const* sf_power_profiles_v2_low_power_cfg_t::p_extend
Pointer to additional settings

◆ p_ioport_pin_tbl_enter

ioport_cfg_t const* sf_power_profiles_v2_low_power_cfg_t::p_ioport_pin_tbl_enter
Pointer to IOPORT settings to apply before entering low power mode

◆ p_ioport_pin_tbl_exit

ioport_cfg_t const* sf_power_profiles_v2_low_power_cfg_t::p_ioport_pin_tbl_exit
Pointer to IOPORT settings to apply after exiting the low power mode

◆ p_lower_lvl_lpm

lpmv2_instance_t const* sf_power_profiles_v2_low_power_cfg_t::p_lower_lvl_lpm
Pointer to an LPMv2 instance

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

ssp_err_t(*	open)(sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_cfg_t const *const p_cfg)
ssp_err_t(*	runApply)(sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_run_cfg_t const *const p_cfg)

ssp_err_t(*	lowPowerApply)(sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_low_power_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_power_profiles_v2_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Framework Power Profiles v2 API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_power_profiles_v2_api_t::close) (sf_power_profiles_v2_ctrl_t *const p_ctrl)

Closes the framework.

Implemented as

- SF_POWER_PROFILES_V2_Close()

Parameters

[in]	p_ctrl	Pointer to control block set in SF_POWER_PROFILES_V2_Open.

◆ lowPowerApply

ssp_err_t(* sf_power_profiles_v2_api_t::lowPowerApply) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_low_power_cfg_t const *const p_cfg)

Applies a Low Power profile.

Implemented as

- SF_POWER_PROFILES_V2_LowPowerApply()

Parameters

[in]	p_ctrl	Pointer to control block set in SF_POWER_PROFILES_V2_Open.
[in]	p_cfg	Pointer to configuration structure. Elements of the structure must be set by user.

◆ open

ssp_err_t(* sf_power_profiles_v2_api_t::open) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_cfg_t const *const p_cfg)

Initializes the framework.

Implemented as

- SF_POWER_PROFILES_V2_Open()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. Elements initialized here.
[in]	p_cfg	Pointer to configuration structure. Elements of the structure must be set by user.

◆ runApply

ssp_err_t(* sf_power_profiles_v2_api_t::runApply) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_run_cfg_t const *const p_cfg)

Applies a Run profile.

Implemented as

- SF_POWER_PROFILES_V2_RunApply()

Parameters

[in]	p_ctrl	Pointer to control block set in SF_POWER_PROFILES_V2_Open.
[in]	p_cfg	Pointer to configuration structure. Elements of the structure must be set by user.

◆ versionGet

ssp_err_t(* sf_power_profiles_v2_api_t::versionGet) (ssp_version_t *const p_version)
Gets version and stores it in provided pointer p_version. **Implemented as** ◦ SF_POWER_PROFILES_V2_VersionGet() **Parameters** [out] p_version — Code and API version used.

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

sf_power_profiles_v2_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Power Profiles V2 Framework Interface

#include <sf_power_profiles_v2_api.h>

Data Fields

sf_power_profiles_v2_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_power_profiles_v2_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_power_profiles_v2_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_power_profiles_v2_api.h

5.1.2.42 SF Socket WIFI Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF Socket WIFI Framework Interface. More...

Data Structures

struct	in_addr
struct	sockaddr
struct	sockaddr_in
struct	sf_socket_ctrl_t
struct	sf_socket_cfg_t
struct	sf_socket_api_t
struct	sf_socket_instance_t

Macros

#define	SF_SOCKET_WIFI_API_VER_MAJOR (2U)
#define	SF_SOCKET_WIFI_API_VER_MINOR (0U)
#define	SF_SOCKET_WIFI_NO_OF_BIT_IN_BYTE (8U)

Typedefs

typedef int32_t	socklen_t

Functions

int	socket (int domain, int type, int protocol) This creates socket for communication. More...

int	close (int sockfd) This closes socket. More...
int	bind (int sockfd, const struct sockaddr *p_local_sock_addr, socklen_t addrlen) This binds socket to IP address. More...
int	listen (int sockfd, int backlog) This listens for connection on socket. More...
int	connect (int sockfd, const struct sockaddr *p_serv_addr, socklen_t addrlen) This connects with remote socket(stream socket). More...
int	accept (int sockfd, struct sockaddr *p_cliaddr, socklen_t *p_addrlen) This accepts connection from remote socket. More...
ssize_t	send (int sockfd, const void *p_buf, size_t length, int flags) This sends data over STREAM socket. More...
ssize_t	recv (int sockfd, void *p_buf, size_t length, int flags) This receives data over STREAM socket. More...
ssize_t	sendto (int sockfd, const void *p_buf, size_t length, int flags, const struct sockaddr *p_dest_addr, socklen_t addrlen) This sends data over DGRAM socket. More...
ssize_t	recvfrom (int sockfd, void *p_buf, size_t length, int flags, struct sockaddr *p_src_addr, socklen_t *p_addrlen) This receives data over DGRAM socket. More...
int	setsockopt (int sockfd, int level, int optname, const void *p_optval, socklen_t optlen) This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...

int	getsockopt (int sockfd, int level, int optname, void *p_optval, socklen_t *p_optlen) This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...
int	select (int nfds, fd_set *p_readfds, fd_set *p_writefds, fd_set *p_exceptfds, struct timeval *p_timeout) This waits for any activity on socket. More...

Detailed Description

RTOS-integrated SF Socket WIFI Framework Interface.

Summary

This SSP Interface provides access OnChip stack BSD Socket API.

Macro Definition Documentation

◆ SF_SOCKET_WIFI_API_VER_MAJOR

#define SF_SOCKET_WIFI_API_VER_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_SOCKET_WIFI_API_VER_MINOR

#define SF_SOCKET_WIFI_API_VER_MINOR (0U)
Minor Version of the API defined in this file

◆ SF_SOCKET_WIFI_NO_OF_BIT_IN_BYTE

#define SF_SOCKET_WIFI_NO_OF_BIT_IN_BYTE (8U)
SF WiFi Socket Number of Bits in a Byte

Typedef Documentation

◆ socklen_t

typedef int32_t socklen_t
Socket Structure Length

Function Documentation

◆ accept()

int accept (int *sockfd*, struct sockaddr * *p_cliaddr*, socklen_t * *p_addrlen*)

This accepts connection from remote socket.

Accept connection request from remote.

Parameters

[in]	sockfd	Local socket
[out]	p_cliaddr	Pointer to remote socket address which trying to connect
[out]	p_addrlen	Pointer to address length of client socket address

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network address
[out]	p_addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error accepting the connection
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Connection is received successfully.

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network

		address (Can be NULL)
[out]	p_addrlen	Size of Socket address structure(Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error accepting the connection or invalid input parameters

Returns

Otherwise Connection is received successfully.

◆ bind()

int bind (int *sockfd*, const struct sockaddr * *p_local_sock_addr*, socklen_t *addrlen*)

This binds socket to IP address.

Bind socket to interface which is identified by IP address

Parameters

[in]	sockfd	Local socket
[in]	p_local_sock_addr	Pointer to local socket address
[in]	addrlen	Size of sock address structure

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Binding socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is bound successfully.

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Binding socket failed or invalid input parameters

Returns

Otherwise Socket is bound successfully.

◆ close()

int close (int *sockfd*)

This closes socket.

API which closes socket

Parameters

[in]	sockfd	Local socket

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Closing socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is closed successfully.

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Closing socket failed or invalid input parameters

Returns

Otherwise Socket is closed successfully.

◆ connect()

int connect (int *sockfd*, const struct sockaddr * *p_serv_addr*, socklen_t *addrlen*)

This connects with remote socket(stream socket).

Establish TCP connection with remote socket

Parameters

[in]	sockfd	Local socket
[in]	p_serv_addr	Pointer to remote socket address

[in]	addrlen	Size of sock address structure

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error occurred.
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is connected successfully.

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error occurred or invalid input parameters

Returns

Otherwise Socket is connected successfully.

◆ getsockopt()

int getsockopt (int *sockfd*, int *level*, int *optname*, void * *p_optval*, socklen_t * *p_optlen*)

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Get Socket options.

Parameters

[in]	sockfd	Local socket
[in]	level	Sockets API level
[in]	optname	Option to be get
[out]	p_optval	Option value to be get
[in]	p_optlen	Length of option value

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error reading socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option read successfully.

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error reading socket option or invalid socket descriptor

Returns

Otherwise Socket option read successfully.

◆ listen()

int listen (int *sockfd*, int *backlog*)

This listens for connection on socket.

Listen for tcp connection. Set socket in listen mode for tcp connection.

Parameters

[in]	sockfd	Local socket
[in]	backlog	Max number of connection queue.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Set socket in Listen mode successfully.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode or invalid input parameters

Returns

Otherwise Set socket in Listen mode successfully.

◆ recv()

ssize_t recv (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*)

This receives data over STREAM socket.

Receive data from remote socket.

Parameters

[in]	sockfd	Local socket
[out]	p_buf	Pointer to data buffer where data will be received
[in]	length	Maximum length of data which can be received
[in]	flags	Socket flags

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to receive data.

Returns

Otherwise Number of Data bytes received successfully.

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[out]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to receive data or invalid input parameters

Returns

Otherwise Data received successfully.

◆ recvfrom()

ssize_t recvfrom (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*, struct sockaddr * *p_src_addr*, socklen_t * *p_addrlen*)

This receives data over DGRAM socket.

Receive data from remote socket.

Parameters

[in]	sockfd	Local socket
[out]	p_buf	Pointer to data buffer where data will be received
[in]	length	Maximum length of data which can be received
[in]	flags	Socket flag
[out]	p_src_addr	Pointer to remote socket address which has sent data
[out]	p_addrlen	Length of socket address structure

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address
[in]	p_addrlen	Size of Remote machine network

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error receiving data

Returns

Otherwise Numbers of data bytes received successfully.

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to

receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address (Can be NULL)
[in]	p_addrlen	Size of Remote machine network (Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error receiving data or invalid input parameters

Returns

Otherwise Numbers of data bytes received successfully.

◆ select()

int select (int *nfds*, fd_set * *p_readfds*, fd_set * *p_writefds*, fd_set * *p_exceptfds*, struct timeval * *p_timeout*)

This waits for any activity on socket.

Wait on a given socket for specified amount of time. In case of any activity e.g. arrival of packet it comes out of wait.

Parameters

[in]	nfds	Max fd
[in]	p_readfds	Pointer to fd_set to check whether data is available for read
[in]	p_writefds	Pointer to fd_set to check whether data is available for write
[in]	p_exceptfds	Pointer to fd_set to check whether exceptional condition occurred
[in]	p_timeout	Wait time in milliseconds

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at

the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set. If no descriptor is to be tested for writing, p_writefds should be NULL
[in]	p_exceptfds	Exceptional socket fd set. If no descriptor is to be tested for exceptions, p_exceptfds should be NULL
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Timeout occurred, no activity.

Returns

Otherwise Activity detected(Packet available).

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set - API does not used this parameter
[in]	p_exceptfds	Exceptional socket fd set - API does not used this parameter
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Timeout occurred, no activity or invalid socket descriptor

Returns

Otherwise Activity detected(Packet available).

◆ send()

ssize_t send (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*)

This sends data over STREAM socket.

Send data to remote socket.

Parameters

[in]	sockfd	Local socket
[in]	p_buf	Pointer to data buffer
[in]	length	Data buffer length
[in]	flags	Socket flags

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to send data.

Returns

Otherwise Number of Data bytes sent successfully.

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to send data or invalid input parameters

Returns

Otherwise Data sent successfully.

◆ sendto()

ssize_t sendto (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*, const struct sockaddr * *p_dest_addr*, socklen_t *addrlen*)

This sends data over DGRAM socket.

Send data to remote socket.

Parameters

[in]	sockfd	Local socket
[in]	p_buf	Pointer to data buffer to sent
[in]	length	Data buffer length
[in]	flags	Socket flag
[in]	p_dest_addr	Pointer to remote socket address where to send data
[in]	addrlen	Length of socket address structure

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data
[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error Sending data.

Returns

Otherwise Numbers of bytes sent successfully.

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data

[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error Sending data or invalid input parameters

Returns

Otherwise Numbers of bytes sent successfully.

◆ setsockopt()

int setsockopt (int *sockfd*, int *level*, int *optname*, const void * *p_optval*, socklen_t *optlen*)

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Set Socket options.

Parameters

[in]	sockfd	Local socket
[in]	level	Sockets API level
[in]	optname	Option to be set
[in]	p_optval	Option value to be set
[in]	optlen	Length of option value

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error setting socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option set successfully.

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error setting socket option or invalid socket descriptor

Returns

Otherwise Socket option set successfully.

◆ socket()

int socket (int *domain*, int *type*, int *protocol*)

This creates socket for communication.

API which creates socket

Parameters

[in]	domain	Socket family
[in]	type	Socket type
[in]	protocol	Protocol type

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket descriptor of newly created socket

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket created successfully

in_addr Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR

Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

unsigned long	s_addr load with inet_aton() More...

Detailed Description

IP address used by sockaddr

Socket Internet Address structure

Field Documentation

◆ s_addr

unsigned long in_addr::s_addr
load with inet_aton() Load with inet_aton()

The documentation for this struct was generated from the following files:

- sf_cellular_socket_api.h
- sf_socket_api.h
- sf_wifi_qca4010_socket_api.h
- nx_bsd.h

sockaddr Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket CELLULAR Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework InterfaceRenesas Synergy Software Package Reference » Framework Interfaces » | SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

short	sin_family Address family.
unsigned short	sin_port Port number.
struct in_addr	sin_addr IP Address.
char	sin_zero [8] zero this if you want to More...

Detailed Description

Socket Address information

Socket Internet Address structure with port and family

Field Documentation

◆ sin_zero

char sockaddr::sin_zero
zero this if you want to Zero this if you want to.

The documentation for this struct was generated from the following files:

- sf_cellular_socket_api.h
- sf_socket_api.h
- sf_wifi_qca4010_socket_api.h
- nx_bsd.h

sf_socket_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_socket_api.h>

Data Fields

sf_wifi_onchip_stack_instance_t *	p_lower_lvl_onchip_wifi low level wifi interface

Detailed Description

Socket Interface control structure

The documentation for this struct was generated from the following file:

- sf_socket_api.h

sf_socket_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_socket_api.h>

Data Fields

sf_wifi_onchip_stack_instance_t *	p_lower_lvl_onchip_wifi Pointer to SF on-chip stack instance.
void *	p_extend Extended configuration.

Detailed Description

Socket Interface configuration structure

The documentation for this struct was generated from the following file:

- sf_socket_api.h

sf_socket_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_socket_api.h>

Data Fields

ssp_err_t(*	open)(sf_socket_ctrl_t *p_ctrl, sf_socket_cfg_t const *const p_cfg) Pointer to function which initializes the network interface for data transfers Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer. More...
ssp_err_t(*	close)(sf_socket_ctrl_t *const p_ctrl) Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Socket Interface API

Field Documentation

◆ close

ssp_err_t(* sf_socket_api_t::close) (sf_socket_ctrl_t *const p_ctrl)

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block

◆ open

ssp_err_t(* sf_socket_api_t::open) (sf_socket_ctrl_t *p_ctrl, sf_socket_cfg_t const *const p_cfg)
Pointer to function which initializes the network interface for data transfers Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer. **Parameters** [in,out] p_ctrl — Pointer to user-provided storage for the control block. [in] p_cfg — Pointer to configuration structure.

◆ versionGet

ssp_err_t(* sf_socket_api_t::versionGet) (ssp_version_t *const p_version)
Gets version and stores it in provided pointer p_version. **Parameters** [out] p_version — pointer to memory location to return version number

---

The documentation for this struct was generated from the following file:

- sf_socket_api.h

sf_socket_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_socket_api.h>

Data Fields

sf_socket_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_socket_cfg_t const *	p_cfg

	Pointer to the configuration structure for this instance.
sf_socket_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_socket_api.h

5.1.2.43 SPI Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SPI Framework Interface. More...

Data Structures

struct	sf_spi_bus_t
struct	sf_spi_cfg_t
struct	sf_spi_api_t
struct	sf_spi_instance_t

Typedefs

typedef void	sf_spi_ctrl_t

Enumerations

enum	sf_spi_dev_state_t { SF_SPI_DEV_STATE_CLOSED = 0, SF_SPI_DEV_STATE_OPENED }

Detailed Description

RTOS-integrated SPI Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SPI Framework. The Interface is implemented by the SPI Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SPI Framework Interface description: SPI Framework

Typedef Documentation

◆ sf_spi_ctrl_t

typedef void sf_spi_ctrl_t
SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. **Implemented as** ◦ sf_spi_instance_ctrl_t

Enumeration Type Documentation

◆ sf_spi_dev_state_t

enum sf_spi_dev_state_t	
SF SPI device state	
Enumerator	
SF_SPI_DEV_STATE_CLOSED	SPI device is closed.
SF_SPI_DEV_STATE_OPENED	SPI device is opened.

sf_spi_bus_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SPI Framework Interface

#include <sf_spi_api.h>

Data Fields

uint8_t	channel

	Channel.
uint32_t	freq_hz_min Bus min frequency supported.
TX_MUTEX *	p_lock_mutex Lock mutex handle for this channel.
TX_MUTEX	device_count_mutex Device count mutex handle for this device.
TX_EVENT_FLAGS_GROUP *	p_sync_eventflag Pointer to the event flag object for SPI data transfer.
sf_spi_ctrl_t **	pp_curr_ctrl Current device using the bus.
uint8_t *	p_bus_name peripheral name SCI_SPI/RSPI
spi_api_t const *	p_lower_lvl_api Pointer to SPI HAL interface to be used in the framework.
uint8_t	device_count Number of devices on the bus, initialize to 0.

Detailed Description

Data structure defining an SPI bus.

The documentation for this struct was generated from the following file:

- sf_spi_api.h

sf_spi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SPI Framework Interface

#include <sf_spi_api.h>

Data Fields

sf_spi_bus_t *	p_bus Bus used by the device.
ioport_port_pin_t	chip_select Chip select for this device.
ioport_level_t	chip_select_level_active Polarity of CS, active High or Low.
spi_cfg_t const *	p_lower_lvl_cfg Pointer to SPI HAL configuration.

Detailed Description

Configuration for Framework SPI driver.

The documentation for this struct was generated from the following file:

- sf_spi_api.h

sf_spi_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SPI Framework Interface

#include <sf_spi_api.h>

Data Fields

ssp_err_t(*	open)(sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)

	Open a designated SPI device on a bus. More...
ssp_err_t(*	read)(sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Receive data from SPI device. More...
ssp_err_t(*	write)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Transmit data to SPI device. More...
ssp_err_t(*	writeRead)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Simultaneously transmit data to an SPI device while receiving data from an SPI device (full duplex). More...
ssp_err_t(*	close)(sf_spi_ctrl_t *const p_ctrl) Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts. More...
ssp_err_t(*	lock)(sf_spi_ctrl_t *const p_ctrl) Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt. More...
ssp_err_t(*	lockWait)(sf_spi_ctrl_t *const p_ctrl, uint32_t const timeout) Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt. The wait option allows thread to wait for the specified timeout when acquiring the bus mutex. More...
ssp_err_t(*	unlock)(sf_spi_ctrl_t *const p_ctrl) Unlock the bus for a particular device and make the bus usable for other devices. More...
ssp_err_t(*	version)(ssp_version_t *const p_version)

Get the version information of the underlying driver. More...

Detailed Description

Definition of the SPI framework interface.

Field Documentation

◆ close

ssp_err_t(* sf_spi_api_t::close) (sf_spi_ctrl_t *const p_ctrl)

Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.

Parameters

[in]	p_ctrl	Pointer to the control block for the device.

◆ lock

ssp_err_t(* sf_spi_api_t::lock) (sf_spi_ctrl_t *const p_ctrl)

Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt.

Parameters

[in]	p_ctrl	Pointer to the control block for the device.

◆ lockWait

ssp_err_t(* sf_spi_api_t::lockWait) (sf_spi_ctrl_t *const p_ctrl, uint32_t const timeout)

Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt. The wait option allows thread to wait for the specified timeout when acquiring the bus mutex.

Implemented as

- SF_SPI_LockWait()

Parameters

[in]	p_ctrl	Pointer to the control block for the device.
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ open

ssp_err_t(* sf_spi_api_t::open) (sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)

Open a designated SPI device on a bus.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to SPI Framework configuration structure.

◆ read

ssp_err_t(* sf_spi_api_t::read) (sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

Receive data from SPI device.

Precondition

Call sf_spi_api_t::open to configure the SPI device before using this function.

Parameters

[in]	p_ctrl	Pointer to the control block for the device.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.
[in]	length	Indicates the number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Indicates data bit width to be transferred.
[in]	timeout	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ unlock

ssp_err_t(* sf_spi_api_t::unlock) (sf_spi_ctrl_t *const p_ctrl)

Unlock the bus for a particular device and make the bus usable for other devices.

Parameters

[in]	p_ctrl	Pointer to the control block for the device.

◆ version

ssp_err_t(* sf_spi_api_t::version) (ssp_version_t *const p_version)

Get the version information of the underlying driver.

Parameters

[out]	p_version	pointer to memory location to return version number

◆ write

ssp_err_t(* sf_spi_api_t::write) (sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

Transmit data to SPI device.

Precondition

Call sf_spi_api_t::open to configure the SPI device before using this function.

Parameters

[in]	p_ctrl	Pointer to the control block for the device.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[in]	length	Indicates the number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Indicates data bit width to be transferred.
[in]	timeout	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

◆ writeRead

ssp_err_t(* sf_spi_api_t::writeRead) (sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

Simultaneously transmit data to an SPI device while receiving data from an SPI device (full duplex).

The writeread API gets mutex object, handles SPI data transmission at SPI HAL layer and receive data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer .

Precondition

Call sf_spi_api_t::open to configure the SPI before using this function.

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.
[in]	length	Indicates the number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Indicates data bit width to be transferred.
[in]	timeout	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

The documentation for this struct was generated from the following file:

- sf_spi_api.h

sf_spi_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SPI Framework Interface

#include <sf_spi_api.h>

Data Fields

sf_spi_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_spi_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_spi_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_spi_api.h

5.1.2.44 Thread Monitor Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Framework Interface for monitoring of threads. More...

Data Structures

struct	sf_thread_monitor_cfg_t

struct	sf_thread_monitor_thread_counter_t
struct	sf_thread_monitor_counter_min_max_t
struct	sf_thread_monitor_api_t
struct	sf_thread_monitor_instance_t

Macros

#define	SF_THREAD_MONITOR_API_VERSION_MAJOR (2U)

Typedefs

typedef void	sf_thread_monitor_ctrl_t

Detailed Description

RTOS-integrated Framework Interface for monitoring of threads.

Any misbehaving threads cause a reset of the device. Both the WDT and IWDT HAL modules are supported by this framework module.

Summary

This is a ThreadX aware Watchdog Timer Thread Monitor Framework for monitoring threads in an application in which threads are executing at an expected rate. Threads to be monitored register themselves through SF_THREAD_MONITOR_ThreadRegister() and increment a count by calling SF_THREAD_MONITOR_CountIncrement() each time they execute. Each monitored thread also provides expected maximum and minimum count values for normal execution.

The Thread Monitor runs periodically and checks the count value of each monitored thread. If the count value falls outside of the expected range of values, the Watchdog Timer is allowed to reset the device. If all thread counts are within their expected ranges, then the Watchdog Timer is refreshed.

The WDT and IWDT modules are supported by the Thread Monitor.

The Framework Layer can be used to protect the entire software project. This is achieved through a high priority thread (Framework Layer) which runs periodically within the refresh permitted window of the Watchdog Timer selected (IWDT is safest as has its own clock source and is started automatically after reset). This thread monitors the state of every other thread in the system. If any of these threads are not running as expected, then the Watchdog Timer is not refreshed and is not allowed to reset the system. If the threads are running as expected, then the Watchdog Timer is refreshed.

Monitoring the other threads is achieved as follows: Each monitored thread increments a count variable each time it runs. The Thread Monitor thread checks the count variable of each thread to make sure it is within an expected range. If any of the variables are out of range a reset is allowed. Otherwise all variables are cleared to zero and the watchdog is refreshed. A profiling mode is used to establish the expected ranges.

This approach is described in the following article:

Jack Ganssle, "Great Watchdog Timers for Embedded Systems," www.ganssle.com/watchdogs.htm

This method requires the instrumenting of each thread to increment its count variable, but this is little overhead for the massive gain in protection.

Interface used: WDT Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Thread Monitor Interface description: Thread Monitor Framework

Macro Definition Documentation

◆ SF_THREAD_MONITOR_API_VERSION_MAJOR

#define SF_THREAD_MONITOR_API_VERSION_MAJOR (2U)
Version of the API defined in this file

Typedef Documentation

◆ sf_thread_monitor_ctrl_t

typedef void sf_thread_monitor_ctrl_t
Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. **Implemented as** ◦ sf_thread_monitor_instance_ctrl_t

sf_thread_monitor_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Thread Monitor Framework Interface

#include <sf_thread_monitor_api.h>

Data Fields

wdt_instance_t const *	p_lower_lvl_wdt Pointer to lower level watchdog instance.

bool	profiling_mode_enabled Enables or disables profiling mode.
UINT	priority Priority of thread monitor thread.

Detailed Description

Configuration for RTOS Thread Monitor driver

The documentation for this struct was generated from the following file:

- sf_thread_monitor_api.h

sf_thread_monitor_thread_counter_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Thread Monitor Framework Interface

#include <sf_thread_monitor_api.h>

Data Fields

uint32_t	current_count Current count value for a thread.
uint32_t	minimum_count
uint32_t	maximum_count
bool	active
TX_THREAD *	p_thread Pointer to thread for this counter data.

Detailed Description

Counter block for each monitored thread.

Field Documentation

◆ active

bool sf_thread_monitor_thread_counter_t::active
Indicates to the monitoring thread whether this count data is currently active. When a thread is registered this value will be set to false as the count is likely to be a partial count and so should not be monitored. This value will be set to true by the thread monitor thread when it clears all counts to zero.

◆ maximum_count

uint32_t sf_thread_monitor_thread_counter_t::maximum_count
Maximum expected count value. If the current count is more than this value the watchdog will reset

◆ minimum_count

uint32_t sf_thread_monitor_thread_counter_t::minimum_count
Minimum expected count value. If the current count is less than this value the watchdog will reset.

The documentation for this struct was generated from the following file:

- sf_thread_monitor_api.h

sf_thread_monitor_counter_min_max_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Thread Monitor Framework Interface

#include <sf_thread_monitor_api.h>

Data Fields

uint32_t	minimum_count
uint32_t	maximum_count

Detailed Description

Counter block for each monitored thread.

Field Documentation

◆ maximum_count

uint32_t sf_thread_monitor_counter_min_max_t::maximum_count
Maximum expected count value. If the current count is more than this value the watchdog will reset

◆ minimum_count

uint32_t sf_thread_monitor_counter_min_max_t::minimum_count
Minimum expected count value. If the current count is less than this value the watchdog will reset.

The documentation for this struct was generated from the following file:

- sf_thread_monitor_api.h

sf_thread_monitor_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Thread Monitor Framework Interface

#include <sf_thread_monitor_api.h>

Data Fields

ssp_err_t(*	open)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg) Configures the WDT or IWDT module. From the configuration data the timeout period of the WDT/IWDT is determined and a thread created monitoring registered threads. More...
ssp_err_t(*	close)(sf_thread_monitor_ctrl_t *const p_ctrl) Suspends the thread monitoring thread. Watchdog peripheral no longer refreshed. More...
ssp_err_t(*	threadRegister)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)

	Registers a thread for monitoring. More...
ssp_err_t(*	threadUnregister)(sf_thread_monitor_ctrl_t *const p_ctrl) Removes a thread from being monitored. More...
ssp_err_t(*	countIncrement)(sf_thread_monitor_ctrl_t *const p_ctrl) Safely increments a monitored thread's count value. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Get version and store it in provided pointer p_version. More...

Detailed Description

Thread monitor API structure. Thread Monitor implementations use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_thread_monitor_api_t::close) (sf_thread_monitor_ctrl_t *const p_ctrl)

Suspends the thread monitoring thread. Watchdog peripheral no longer refreshed.

Implemented as

- SF_THREAD_MONITOR_Close()

Parameters

[in,out]	p_ctrl	Control structure set in SF_THREAD_MONITOR_Open.

◆ countIncrement

ssp_err_t(* sf_thread_monitor_api_t::countIncrement) (sf_thread_monitor_ctrl_t *const p_ctrl)

Safely increments a monitored thread's count value.

Implemented as

- SF_THREAD_MONITOR_CountIncrement()

Parameters

[in,out]	p_ctrl	Control structure set in SF_THREAD_MONITOR_Open.

◆ open

ssp_err_t(* sf_thread_monitor_api_t::open) (sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg)

Configures the WDT or IWDT module. From the configuration data the timeout period of the WDT/IWDT is determined and a thread created monitoring registered threads.

Implemented as

- SF_THREAD_MONITOR_Open()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. Elements initialized here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ threadRegister

ssp_err_t(* sf_thread_monitor_api_t::threadRegister) (sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)

Registers a thread for monitoring.

Implemented as

- SF_THREAD_MONITOR_ThreadRegister()

Parameters

[in,out]	p_ctrl	Control structure set in SF_THREAD_MONITOR_Open.
[in]	p_counter_min_max	Pointer to structure containing min and max values for thread to be registered values.

◆ threadUnregister

ssp_err_t(* sf_thread_monitor_api_t::threadUnregister) (sf_thread_monitor_ctrl_t *const p_ctrl)

Removes a thread from being monitored.

Implemented as

- SF_THREAD_MONITOR_ThreadUnregister()

Parameters

[in,out]	p_ctrl	Control structure set in SF_THREAD_MONITOR_Open.

◆ versionGet

ssp_err_t(* sf_thread_monitor_api_t::versionGet) (ssp_version_t *const p_version)

Get version and store it in provided pointer p_version.

Implemented as

- SF_THREAD_MONITOR_VersionGet()

Parameters

[in,out]	p_version	Pointer to structure storing API and code versions.

The documentation for this struct was generated from the following file:

- sf_thread_monitor_api.h

sf_thread_monitor_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Thread Monitor Framework Interface

#include <sf_thread_monitor_api.h>

Data Fields

sf_thread_monitor_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_thread_monitor_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_thread_monitor_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_thread_monitor_api.h

5.1.2.45 CTSU v2 Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

CTSU v2 Framework Interface. More...

Data Structures

struct	sf_touch_ctsu_button_cfg_t
struct	sf_touch_ctsu_slider_cfg_t
struct	sf_touch_ctsu_wheel_cfg_t
struct	sf_touch_ctsu_cfg_t
struct	sf_touch_ctsu_api_t
struct	sf_touch_ctsu_instance_t

Typedefs

typedef void	sf_touch_ctsu_ctrl_t
typedef struct st_ctsu_callback_args	sf_touch_ctsu_callback_args_t

Detailed Description

CTSU v2 Framework Interface.

Summary

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CTSU v2 Framework Interface description: Capacitive Touch v2 Framework

Typedef Documentation

◆ sf_touch_ctsu_callback_args_t

typedef struct st_ctsu_callback_args sf_touch_ctsu_callback_args_t
Callback function parameter data

◆ sf_touch_ctsu_ctrl_t

typedef void sf_touch_ctsu_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ sf_touch_ctsu_instance_ctrl_t

sf_touch_ctsu_button_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

uint8_t	elem_index Element number used by this button.
uint16_t	threshold Touch/non-touch judgment threshold.
uint16_t	hysteresis Threshold hysteresis for chattering prevention.

Detailed Description

Configuration of each button

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

sf_touch_ctsu_slider_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework

Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

uint8_t const *	p_elem_index Element number array used by this slider.
uint8_t	num_elements Number of elements used by this slider.
uint16_t	threshold Position calculation start threshold value.

Detailed Description

Configuration of each slider

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

sf_touch_ctsu_wheel_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

uint8_t const *	p_elem_index Element number array used by this wheel.
uint8_t	num_elements Number of elements used by this wheel.

uint16_t	threshold Position calculation start threshold value.

Detailed Description

Configuration of each wheel

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

sf_touch_ctsu_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

sf_touch_ctsu_button_cfg_t const *	p_buttons Pointer to array of button configuration.
sf_touch_ctsu_slider_cfg_t const *	p_sliders Pointer to array of slider configuration.
sf_touch_ctsu_wheel_cfg_t const *	p_wheels Pointer to array of wheel configuration.
uint8_t	num_buttons Number of buttons.
uint8_t	num_sliders Number of sliders.

uint8_t	num_wheels Number of wheels.
uint8_t	on_freq The cumulative number of determinations of ON.
uint8_t	off_freq The cumulative number of determinations of OFF.
uint16_t	drift_freq Base value drift frequency. [0 : no use].
uint16_t	cancel_freq Maximum continuous ON. [0 : no use].
uint8_t	number Configuration number for QE monitor.
ctsu_instance_t const *	p_ctsu_instance Pointer to CTSU instance.
void *	p_uart_instance Pointer to UART instance.
void const *	p_context User defined context passed into callback function.
void const *	p_extend Pointer to extended configuration by instance of interface.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

sf_touch_ctsu_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

ssp_err_t(*	open)(sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)
ssp_err_t(*	scanStart)(sf_touch_ctsu_ctrl_t *const p_ctrl)
ssp_err_t(*	dataGet)(sf_touch_ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)
ssp_err_t(*	callbackSet)(sf_touch_ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(sf_touch_ctsu_callback_args_t *), void const *const p_context, sf_touch_ctsu_callback_args_t *const p_callback_memory)
ssp_err_t(*	close)(sf_touch_ctsu_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_data)

Detailed Description

Functions implemented at the HAL layer will follow this API.

Field Documentation

◆ callbackSet

ssp_err_t(* sf_touch_ctsu_api_t::callbackSet) (sf_touch_ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(sf_touch_ctsu_callback_args_t *), void const *const p_context, sf_touch_ctsu_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- SF_TOUCH_CTSU_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the CTSU control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

ssp_err_t(* sf_touch_ctsu_api_t::close) (sf_touch_ctsu_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- SF_TOUCH_CTSU_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ dataGet

ssp_err_t(* sf_touch_ctsu_api_t::dataGet) (sf_touch_ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)

Data get.

Implemented as

- SF_TOUCH_CTSU_DataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_button_status	Pointer to get data bitmap.
[out]	p_slider_position	Pointer to get data array.
[out]	p_wheel_position	Pointer to get data array.

◆ open

ssp_err_t(* sf_touch_ctsu_api_t::open) (sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)

Open driver.

Implemented as

- SF_TOUCH_CTSU_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ scanStart

ssp_err_t(* sf_touch_ctsu_api_t::scanStart) (sf_touch_ctsu_ctrl_t *const p_ctrl)

Scan start.

Implemented as

- SF_TOUCH_CTSU_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ versionGet

ssp_err_t(* sf_touch_ctsu_api_t::versionGet) (ssp_version_t *const p_data)

Return the version of the driver.

Implemented as

- SF_TOUCH_CTSU_VersionGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

sf_touch_ctsu_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » CTSU v2 Framework Interface

#include <sf_touch_ctsuv2_api.h>

Data Fields

sf_touch_ctsu_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_touch_ctsu_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_touch_ctsu_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2_api.h

5.1.2.46 Touch chip Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Touch chip Interface. More...

Data Structures

struct	sf_touch_panel_chip_cfg_t
struct	sf_touch_panel_chip_api_t
struct	sf_touch_panel_chip_instance_t

Typedefs

typedef void	sf_touch_panel_chip_ctrl_t

Detailed Description

RTOS-integrated Touch chip Interface.

Summary

This module is a ThreadX-aware Touch chip interface which gets the data from the touch chip. This Interface is implemented by Touch Panel V2 Framework.

Interfaces used:

- Touch Panel Framework Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Touch Panel chip Framework Interface description: Touch Panel V2 Framework

Typedef Documentation

◆ sf_touch_panel_chip_ctrl_t

typedef void sf_touch_panel_chip_ctrl_t

Touch panel chip framework control block. Allocate an instance specific control block to pass into the touch panel chip framework API calls.

Implemented as

- sf_touch_panel_chip_sx8654_instance_ctrl_t
- sf_touch_panel_chip_ft5x06_instance_ctrl_t

sf_touch_panel_chip_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch chip Interface

#include <sf_touch_panel_chip_api.h>

Detailed Description

Configuration for RTOS integrated touch driver.

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_api.h

sf_touch_panel_chip_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch chip Interface

#include <sf_touch_panel_chip_api.h>

Data Fields

ssp_err_t(*	open)(sf_touch_panel_chip_ctrl_t *const p_ctrl, sf_touch_panel_chip_cfg_t const *const p_cfg) Initializes the touch chip. More...
ssp_err_t(*	payloadGet)(sf_touch_panel_chip_ctrl_t *const p_ctrl, sf_touch_panel_v2_payload_t *p_payload) Reads the touch chip and fills in the touch payload data. More...

ssp_err_t(*	reset)(sf_touch_panel_chip_ctrl_t *const p_ctrl) Resets the touch chip. More...
ssp_err_t(*	close)(sf_touch_panel_chip_ctrl_t *const p_ctrl) Close the touch chip. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets the chip version and stores it in provided pointer p_version. More...

Detailed Description

Touch panel chip API structure. Touch panel chip implementations use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_touch_panel_chip_api_t::close) (sf_touch_panel_chip_ctrl_t *const p_ctrl)

Close the touch chip.

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.

◆ open

ssp_err_t(* sf_touch_panel_chip_api_t::open) (sf_touch_panel_chip_ctrl_t *const p_ctrl, sf_touch_panel_chip_cfg_t const *const p_cfg)

Initializes the touch chip.

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ payloadGet

ssp_err_t(* sf_touch_panel_chip_api_t::payloadGet) (sf_touch_panel_chip_ctrl_t *const p_ctrl, sf_touch_panel_v2_payload_t *p_payload)

Reads the touch chip and fills in the touch payload data.

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.
[out]	p_payload	Pointer to the payload data structure. Touch data provided should be processed to logical pixel values.

◆ reset

ssp_err_t(* sf_touch_panel_chip_api_t::reset) (sf_touch_panel_chip_ctrl_t *const p_ctrl)

Resets the touch chip.

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.

◆ versionGet

ssp_err_t(* sf_touch_panel_chip_api_t::versionGet) (ssp_version_t *const p_version)

Gets the chip version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used are stored here.

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_api.h

sf_touch_panel_chip_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch chip Interface

#include <sf_touch_panel_chip_api.h>

Data Fields

sf_touch_panel_chip_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_touch_panel_chip_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.

sf_touch_panel_chip_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_api.h

5.1.2.47 Touch Panel Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated Touch Panel Framework Interface. More...

Data Structures

struct	sf_touch_panel_v2_payload_t
struct	sf_touchpanel_v2_callback_args_t
struct	sf_touch_panel_v2_cfg_t
struct	sf_touch_panel_v2_calibrate_t
struct	sf_touch_panel_v2_calibrate_factors_t
struct	sf_touch_panel_v2_api_t
struct	sf_touch_panel_v2_instance_t

Typedefs

typedef void	sf_touch_panel_v2_ctrl_t

Enumerations

enum	sf_touch_panel_v2_event_t { SF_TOUCH_PANEL_V2_EVENT_INVALID, SF_TOUCH_PANEL_V2_EVENT_HOLD, SF_TOUCH_PANEL_V2_EVENT_MOVE,

```
                                SF_TOUCH_PANEL_V2_EVENT_DOWN,
                                 SF_TOUCH_PANEL_V2_EVENT_UP,
                                SF_TOUCH_PANEL_V2_EVENT_NONE
                                }
```

Detailed Description

RTOS-integrated Touch Panel Framework Interface.

Summary

This module is a ThreadX-aware Touch Panel V2 Framework which scans for touch events and posts them to the user callback. This Interface is implemented by Touch Panel V2 Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Touch Panel V2 Framework Interface description: Touch Panel V2 Framework

Typedef Documentation

◆ sf_touch_panel_v2_ctrl_t

typedef void sf_touch_panel_v2_ctrl_t

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls.

Implemented as

- sf_touch_panel_v2_instance_ctrl_t

Enumeration Type Documentation

◆ sf_touch_panel_v2_event_t

enum sf_touch_panel_v2_event_t	
Touch event list.	
Enumerator	
SF_TOUCH_PANEL_V2_EVENT_INVALID	Invalid touch data.
SF_TOUCH_PANEL_V2_EVENT_HOLD	Touch has not moved since last touch event.
SF_TOUCH_PANEL_V2_EVENT_MOVE	Touch has moved since last touch event.
SF_TOUCH_PANEL_V2_EVENT_DOWN	New touch event reported.
SF_TOUCH_PANEL_V2_EVENT_UP	Touch released.
SF_TOUCH_PANEL_V2_EVENT_NONE	No valid touch event happened.

sf_touch_panel_v2_payload_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

int16_t	x X coordinate.
int16_t	y Y coordinate.
sf_touch_panel_v2_event_t	event_type Touch event type.

Detailed Description

Touch data payload

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touchpanel_v2_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

sf_touch_panel_v2_payload_t	payload Touch data and event provided to the user during callback.
void const *	p_context Context provided to user during callback.

Detailed Description

User callback

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touch_panel_v2_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

uint16_t	hsize_pixels Horizontal size of screen in pixels.

uint16_t	vsize_pixels Vertical size of screen in pixels.
UINT	priority Priority of the touch panel thread.
uint16_t	update_hz
uint16_t	rotation_angle Touch coordinate rotation angle(0/90/180/270)
void const *	p_extend
void const *	p_context User defined context passed into callback function.

Detailed Description

Configuration for RTOS integrated touch panel framework.

Field Documentation

◆ p_extend

void const* sf_touch_panel_v2_cfg_t::p_extend
Pointer to hardware specific extension.

◆ update_hz

uint16_t sf_touch_panel_v2_cfg_t::update_hz
The frequency to report repeat (SF_TOUCH_PANEL_V2_EVENT_DOWN or SF_TOUCH_PANEL_V2_EVENT_HOLD) touch events in Hertz. ***Note*** *This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.*

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touch_panel_v2_calibrate_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

int32_t	x Expected x coordinate.
int32_t	y Expected y coordinate.
void const *	p_extend

Detailed Description

Calibration data passed to SF_TOUCH_PANEL_V2_Calibrate.

Field Documentation

◆ p_extend

void const* sf_touch_panel_v2_calibrate_t::p_extend
Pointer to hardware specific extension.

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touch_panel_v2_calibrate_factors_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Detailed Description

Calibration factors calculated in order to calibrate the touch data.

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touch_panel_v2_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

ssp_err_t(*	open)(sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_cfg_t const *const p_cfg) Create required RTOS objects, call lower level module for hardware specific initialization, and create a thread to post touch data to user application. More...
ssp_err_t(*	calibrate)(sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_calibrate_t const *const p_display, sf_touch_panel_v2_calibrate_t const *const p_touchscreen, ULONG const timeout) Begin calibration routine based on provided expected and actual coordinates. More...
ssp_err_t(*	start)(sf_touch_panel_v2_ctrl_t *const p_ctrl) Start scanning for touch events. More...
ssp_err_t(*	touchDataGet)(sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_payload_t *p_payload, ULONG const timeout) Reads the touch data and fills in the touch payload data. More...
ssp_err_t(*	stop)(sf_touch_panel_v2_ctrl_t *const p_ctrl) Stop scanning for touch events. More...

ssp_err_t(*	reset)(sf_touch_panel_v2_ctrl_t *const p_ctrl) Reset touch chip if reset pin is provided. More...
ssp_err_t(*	close)(sf_touch_panel_v2_ctrl_t *const p_ctrl) Terminate touch thread and close channel at HAL layer. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Touch panel V2 API structure. Touch panel V2 implementations use the following API.

Field Documentation

◆ calibrate

ssp_err_t(* sf_touch_panel_v2_api_t::calibrate) (sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_calibrate_t const *const p_display, sf_touch_panel_v2_calibrate_t const *const p_touchscreen, ULONG const timeout)

Begin calibration routine based on provided expected and actual coordinates.

Implemented as

- SF_TOUCH_PANEL_V2_Calibrate()

Parameters

[in]	p_ctrl	Handle set in sf_touch_panel_v2_api_t::open.
[in]	p_display	Expected coordinates of the display.
[in]	p_touchscreen	Actual coordinates obtained from the touch driver.
[in]	timeout	ThreadX timeout. Select TX_NO_WAIT, a value in system clock counts between 1 and 0xFFFFFFFF, or TX_WAIT_FOREVER.

◆ close

ssp_err_t(* sf_touch_panel_v2_api_t::close) (sf_touch_panel_v2_ctrl_t *const p_ctrl)

Terminate touch thread and close channel at HAL layer.

Implemented as

- SF_TOUCH_PANEL_V2_Close()

Parameters

[in]	p_ctrl	Handle set in sf_touch_panel_v2_api_t::open.

◆ open

ssp_err_t(* sf_touch_panel_v2_api_t::open) (sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_cfg_t const *const p_cfg)

Create required RTOS objects, call lower level module for hardware specific initialization, and create a thread to post touch data to user application.

Implemented as

- SF_TOUCH_PANEL_V2_Open()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ reset

ssp_err_t(* sf_touch_panel_v2_api_t::reset) (sf_touch_panel_v2_ctrl_t *const p_ctrl)

Reset touch chip if reset pin is provided.

Implemented as

- SF_TOUCH_PANEL_V2_Reset()

Note

This does not include calibration. Use sf_touch_panel_v2_api_t::calibrate from the application after this function if calibration is required after reset.

Parameters

[in]	p_ctrl	Handle set in sf_touch_panel_v2_api_t::open.

◆ start

ssp_err_t(* sf_touch_panel_v2_api_t::start) (sf_touch_panel_v2_ctrl_t *const p_ctrl)

Start scanning for touch events.

Implemented as

- SF_TOUCH_PANEL_V2_Start()

Parameters

[in]	p_ctrl	Handle set in sf_touch_panel_v2_api_t::open.

◆ stop

ssp_err_t(* sf_touch_panel_v2_api_t::stop) (sf_touch_panel_v2_ctrl_t *const p_ctrl)

Stop scanning for touch events.

Implemented as

- SF_TOUCH_PANEL_V2_Stop()

Parameters

[in]	p_ctrl	Handle set in sf_touch_panel_v2_api_t::open.

◆ touchDataGet

ssp_err_t(* sf_touch_panel_v2_api_t::touchDataGet) (sf_touch_panel_v2_ctrl_t *const p_ctrl, sf_touch_panel_v2_payload_t *p_payload, ULONG const timeout)

Reads the touch data and fills in the touch payload data.

Implemented as

- SF_TOUCH_PANEL_V2_TouchDataGet()

Parameters

[in,out]	p_ctrl	Pointer to a structure allocated by user. This control structure is initialized in this function.
[out]	p_payload	Pointer to the payload to data structure. Touch data provided should be processed to logical pixel values.
[in]	timeout	ThreadX timeout. Select TX_NO_WAIT, a value in system clock counts between 1 and 0xFFFFFFFF, or TX_WAIT_FOREVER.

◆ versionGet

ssp_err_t(* sf_touch_panel_v2_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Implemented as

- SF_TOUCH_PANEL_V2_VersionGet()

Parameters

[out]	p_version	Code and API version used stored here.

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

sf_touch_panel_v2_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » Touch Panel Framework Interface

#include <sf_touch_panel_v2_api.h>

Data Fields

sf_touch_panel_v2_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_touch_panel_v2_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_touch_panel_v2_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2_api.h

5.1.2.48 SF WIFI Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF WIFI Framework Interface. More...

Data Structures

struct	sf_wifi_ip_addr_t
struct	sf_wifi_info_t

struct	sf_wifi_callback_args_t
struct	sf_wifi_provisioning_t
struct	sf_wifi_cfg_t
struct	sf_wifi_stats_t
struct	sf_wifi_scan_t
struct	sf_wifi_wps_t
struct	sf_wifi_ctrl_t
struct	sf_wifi_api_t
struct	sf_wifi_instance_t

Macros

#define	SF_WIFI_API_VERSION_MAJOR (2U)
#define	SF_WIFI_API_VERSION_MINOR (0U)
#define	SF_WIFI_SSID_LENGTH (32U) WiFi SSID length.
#define	SF_WIFI_SECURITY_KEY_LENGTH (128U) WiFi Security Key length.
#define	SF_WIFI_MAC_ADDR_LENGTH (6U) WiFi MAC address length.
#define	SF_WIFI_WPS_PIN_LENGTH (8U) WiFi WPS Pin length.
#define	SF_WIFI_TRUE (1U) Boolean True condition.
#define	SF_WIFI_FALSE (0U) Boolean False condition.

#define	SF_WIFI_NULL_BYTE ((uint8_t)'\0') NULL Byte.
#define	SF_WIFI_SIZE_FOR_NULL_BYTE (1U) Size in byte for NULL.
#define	SF_WIFI_IPV4_ADDRESS(a, b, c, d)

Enumerations

enum	sf_wifi_ip_addr_version_t { SF_WIFI_IP_ADDR_VERSION_4, SF_WIFI_IP_ADDR_VERSION_6 }
enum	sf_wifi_interface_mode_t { SF_WIFI_INTERFACE_MODE_AP, SF_WIFI_INTERFACE_MODE_CLIENT }
enum	sf_wifi_wep_key_format_t { SF_WIFI_WEP_KEY_FORMAT_ASCII, SF_WIFI_WEP_KEY_FORMAT_HEX }
enum	sf_wifi_security_type_t { SF_WIFI_SECURITY_TYPE_OPEN, SF_WIFI_SECURITY_TYPE_WEP, SF_WIFI_SECURITY_TYPE_WPA, SF_WIFI_SECURITY_TYPE_WPA2 }
enum	sf_wifi_encryption_type_t { SF_WIFI_ENCRYPTION_TYPE_AUTO, SF_WIFI_ENCRYPTION_TYPE_TKIP, SF_WIFI_ENCRYPTION_TYPE_CCMP, SF_WIFI_ENCRYPTION_TYPE_WEP, SF_WIFI_ENCRYPTION_TYPE_NONE }
enum	sf_wifi_bss_type_t { SF_WIFI_BSS_TYPE_INFRASTRUCTURE = 0, SF_WIFI_BSS_TYPE_ADHOC = 1, SF_WIFI_BSS_TYPE_ANY = 2, SF_WIFI_BSS_TYPE_UNKNOWN = -1 }
enum	sf_wifi_interface_hw_mode_t { SF_WIFI_INTERFACE_HW_MODE_11A, SF_WIFI_INTERFACE_HW_MODE_11B, SF_WIFI_INTERFACE_HW_MODE_11G, SF_WIFI_INTERFACE_HW_MODE_11N }
enum	sf_wifi_rts_t { SF_WIFI_RTS_DISABLE, SF_WIFI_RTS_ENABLE }
enum	sf_wifi_preamble_t { SF_WIFI_PREAMBLE_SHORT, SF_WIFI_PREAMBLE_LONG }
enum	sf_wifi_wmm_t { SF_WIFI_WMM_DISABLE, SF_WIFI_WMM_ENABLE }
enum	sf_wifi_high_throughput_t { SF_WIFI_HIGH_THROUGHPUT_DISABLE, SF_WIFI_HIGH_THROUGHPUT_ENABLE }

enum	sf_wifi_ssid_broadcast_t { SF_WIFI_SSID_BROADCAST_DISABLE, SF_WIFI_SSID_BROADCAST_ENABLE }
enum	sf_wifi_wds_t { SF_WIFI_WDS_DISABLE, SF_WIFI_WDS_ENABLE }
enum	sf_wifi_mandatory_high_throughput_t { SF_WIFI_MANDATORY_HIGH_THROUGHPUT_DISABLE, SF_WIFI_MANDATORY_HIGH_THROUGHPUT_ENABLE }
enum	sf_wifi_auto_negotiation_t { SF_WIFI_AUTO_NEGOTIATION_DISABLE, SF_WIFI_AUTO_NEGOTIATION_ENABLE }
enum	sf_wifi_access_control_t { SF_WIFI_ACCESS_CONTROL_DISABLE, SF_WIFI_ACCESS_CONTROL_DENY, SF_WIFI_ACCESS_CONTROL_ALLOW }
enum	sf_wifi_wps_mode_t { SF_WIFI_WPS_MODE_PUSHBUTTON, SF_WIFI_WPS_MODE_PIN }
enum	sf_wifi_event_t { SF_WIFI_EVENT_RX = (1 << 0), SF_WIFI_EVENT_AP_CONNECT = (1 << 1), SF_WIFI_EVENT_AP_DISCONNECT = (1 << 2), SF_WIFI_EVENT_CLIENT_CONNECT = (1 << 3), SF_WIFI_EVENT_CLIENT_DISCONNECT = (1 << 4) }

Detailed Description

RTOS-integrated SF WIFI Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI Framework.

Macro Definition Documentation

◆ SF_WIFI_API_VERSION_MAJOR

#define SF_WIFI_API_VERSION_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_WIFI_API_VERSION_MINOR

#define SF_WIFI_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

◆ SF_WIFI_IPV4_ADDRESS

```
#define SF_WIFI_IPV4_ADDRESS ( a, b, c, d  )
((((uint32_t) a) << (24U)) | (((uint32_t) b) << (16U)) | \
                                        (((uint32_t) c) << (8U)) | ((uint32_t) d))
```

IP Address Generation Macro

Enumeration Type Documentation

◆ sf_wifi_access_control_t

enum sf_wifi_access_control_t

WiFi Framework AccessContol mode

Enumerator	
SF_WIFI_ACCESS_CONTROL_DISABLE	Disable MAC address matching.
SF_WIFI_ACCESS_CONTROL_DENY	Deny association to stations on the MAC list.
SF_WIFI_ACCESS_CONTROL_ALLOW	Allow association to stations on the MAC list.

◆ sf_wifi_auto_negotiation_t

enum sf_wifi_auto_negotiation_t

WiFi Auto Negotiation flag

Enumerator	
SF_WIFI_AUTO_NEGOTIATION_DISABLE	Auto negotiation disable.
SF_WIFI_AUTO_NEGOTIATION_ENABLE	Auto negotiation enable.

◆ sf_wifi_bss_type_t

enum sf_wifi_bss_type_t	
WiFi BSS type	
Enumerator	
SF_WIFI_BSS_TYPE_INFRASTRUCTURE	Infrastructure network.
SF_WIFI_BSS_TYPE_ADHOC	802.11 ad-hoc IBSS network
SF_WIFI_BSS_TYPE_ANY	Either infrastructure or ad-hoc network.
SF_WIFI_BSS_TYPE_UNKNOWN	BSS type is unknown.

◆ sf_wifi_encryption_type_t

enum sf_wifi_encryption_type_t	
WiFi Encryption type	
Enumerator	
SF_WIFI_ENCRYPTION_TYPE_AUTO	Automatic selection of encryption protocol.
SF_WIFI_ENCRYPTION_TYPE_TKIP	Temporal Key Integrity Protocol. Used by WPA.
SF_WIFI_ENCRYPTION_TYPE_CCMP	CTR mode with CBC-MAC Protocol. Used by WPA2.
SF_WIFI_ENCRYPTION_TYPE_WEP	WEP mode. Used by WEP.
SF_WIFI_ENCRYPTION_TYPE_NONE	No Encryption. Used by Open Security type.

◆ sf_wifi_event_t

enum sf_wifi_event_t	
WiFi Framework event codes	
Enumerator	
SF_WIFI_EVENT_RX	Packet received event.
SF_WIFI_EVENT_AP_CONNECT	Device Associated Successfully with AP.
SF_WIFI_EVENT_AP_DISCONNECT	Device Disconnected with AP.
SF_WIFI_EVENT_CLIENT_CONNECT	Client Associated Successfully with device AP.
SF_WIFI_EVENT_CLIENT_DISCONNECT	Client Disconnected from device AP.

◆ sf_wifi_high_throughput_t

enum sf_wifi_high_throughput_t	
WiFi High Throughput flag	
Enumerator	
SF_WIFI_HIGH_THROUGHPUT_DISABLE	Disable high throughput mode.
SF_WIFI_HIGH_THROUGHPUT_ENABLE	Enable high throughput mode. Also requires WMM to be enabled.

◆ sf_wifi_interface_hw_mode_t

enum sf_wifi_interface_hw_mode_t	
WiFi Hardware mode	
Enumerator	
SF_WIFI_INTERFACE_HW_MODE_11A	802.11a
SF_WIFI_INTERFACE_HW_MODE_11B	802.11b
SF_WIFI_INTERFACE_HW_MODE_11G	802.11g
SF_WIFI_INTERFACE_HW_MODE_11N	802.11n

◆ sf_wifi_interface_mode_t

enum sf_wifi_interface_mode_t	
WiFi Interface mode	
Enumerator	
SF_WIFI_INTERFACE_MODE_AP	Access Point mode.
SF_WIFI_INTERFACE_MODE_CLIENT	Station Mode.

◆ sf_wifi_ip_addr_version_t

enum sf_wifi_ip_addr_version_t	
IP address version	
Enumerator	
SF_WIFI_IP_ADDR_VERSION_4	IPv4 address.
SF_WIFI_IP_ADDR_VERSION_6	IPv6 address.

◆ sf_wifi_mandatory_high_throughput_t

enum sf_wifi_mandatory_high_throughput_t	
WiFi Mandatory High Throughput flag	
Enumerator	
SF_WIFI_MANDATORY_HIGH_THROUGHPUT_DISABLE	Disable Mandatory HT requirement.
SF_WIFI_MANDATORY_HIGH_THROUGHPUT_ENABLE	Enable mandatory HT requirement.

◆ sf_wifi_preamble_t

enum sf_wifi_preamble_t	
WiFi Preamble type	
Enumerator	
SF_WIFI_PREAMBLE_SHORT	Use short preamble.
SF_WIFI_PREAMBLE_LONG	Use long preamble.

◆ sf_wifi_rts_t

enum sf_wifi_rts_t	
WiFi RTS flag	
Enumerator	
SF_WIFI_RTS_DISABLE	Disable RTS/CTS handshake.
SF_WIFI_RTS_ENABLE	Enable RTS/CTS handshake.

◆ sf_wifi_security_type_t

enum sf_wifi_security_type_t	
WiFi Security type	
Enumerator	
SF_WIFI_SECURITY_TYPE_OPEN	Open. No encryption used.
SF_WIFI_SECURITY_TYPE_WEP	128-bit WEP OPEN ASCII
SF_WIFI_SECURITY_TYPE_WPA	WiFi Protected Access.
SF_WIFI_SECURITY_TYPE_WPA2	WiFi Protected Access v2.

◆ sf_wifi_ssid_broadcast_t

enum sf_wifi_ssid_broadcast_t	
WiFi SSID Broadcast flag	
Enumerator	
SF_WIFI_SSID_BROADCAST_DISABLE	Disable SSID Broadcast.
SF_WIFI_SSID_BROADCAST_ENABLE	Enable SSID Broadcast.

◆ sf_wifi_wds_t

enum sf_wifi_wds_t	
WiFi WDS Flasg	
Enumerator	
SF_WIFI_WDS_DISABLE	Disable WDS.
SF_WIFI_WDS_ENABLE	Enable WDS.

◆ sf_wifi_wep_key_format_t

enum sf_wifi_wep_key_format_t	
WiFi WEP Key Format	
Enumerator	
SF_WIFI_WEP_KEY_FORMAT_ASCII	WEP Key in ASCII.
SF_WIFI_WEP_KEY_FORMAT_HEX	WEP Key in Hex.

◆ sf_wifi_wmm_t

enum sf_wifi_wmm_t	
WiFi WMM flag	
Enumerator	
SF_WIFI_WMM_DISABLE	Disable WMM.
SF_WIFI_WMM_ENABLE	Enable WMM.

◆ sf_wifi_wps_mode_t

enum sf_wifi_wps_mode_t	
WiFi WPS mode	
Enumerator	
SF_WIFI_WPS_MODE_PUSHBUTTON	WPS Push button method.
SF_WIFI_WPS_MODE_PIN	WPS pin method.

sf_wifi_ip_addr_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_ip_addr_version_t	version IP Address Version : v4 or v6.
union {	
}	addr IP address.

Detailed Description

IP address information

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

uint8_t *	p_chipset Pointer to sting showing WiFi chipset/driver information.
uint16_t	rssi Received signal strength indication.

uint16_t	noise_level Signal to noise ratio.
uint16_t	link_quality Signal strength / quality.

Detailed Description

Configuration about underlying device driver.

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_event_t	event Event code.
uint8_t *	p_data Packet data.
uint32_t	length Packet Data length.
uint8_t	mac_addr [SF_WIFI_MAC_ADDR_LENGTH] Client station MAC address.
void const *	p_context

	Context provided to user during callback.

Detailed Description

WiFi framework callback parameter definition

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_provisioning_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_interface_mode_t	mode Select AP or Client mode.
uint8_t	channel RF Channel number.
uint8_t	ssid [SF_WIFI_SSID_LENGTH+1] SSID.
sf_wifi_security_type_t	security Security type.
sf_wifi_encryption_type_t	encryption Encryption type.
uint8_t	key [SF_WIFI_SECURITY_KEY_LENGTH] Pre-shared key.

void(*	p_callback)(sf_wifi_callback_args_t *p_args) Pointer to Connection status notification callback function.

Detailed Description

WiFi Provisioning parameters

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

uint8_t	mac_addr [6] MAC address of WiFi Device.
sf_wifi_interface_hw_mode_t	hw_mode Modulation type: 11a/b/g/n.
uint8_t	tx_power Sets transmit power in dBm.
sf_wifi_rts_t	rts RTS/CTS handshake flag.
uint16_t	fragmentation Fragmentation threshold.

uint8_t	dtim Delivery traffic indication message interval.
sf_wifi_high_throughput_t	high_throughput High-throughput mode. Only valid for 802.11n.
sf_wifi_preamble_t	preamble Preamble type.
sf_wifi_wmm_t	wmm WiFi Multimedia Mode flag. If enabled, also requires.
uint8_t	max_stations Maximum permitted stations. Valid in AP mode only.
sf_wifi_ssid_broadcast_t	ssid_broadcast SSID broadcast flag. Valid in AP mode only.
sf_wifi_access_control_t	access_control Mode of access control MAC list.
uint32_t	beacon Beacon interval. Valid in AP mode only.
uint32_t	station_inactivity_timeout Station inactivity timeout value. Valid in AP mode only.
sf_wifi_wds_t	wds WDS flag. Valid in AP mode only.
void *	p_buffer_pool_rx Pointer to Network stack Rx buffer pool.
sf_wifi_mandatory_high_thro	req_high_throughput

ughput_t	Only allow HT mode. Valid in AP mode only.
void(*	p_callback)(sf_wifi_callback_args_t *p_args) Pointer to callback function.
void const *	p_context User defined context passed into callback function.
void const *	p_extend Instance specific configuration.

Detailed Description

Define the WiFi configuration parameters

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_stats_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

uint32_t	rx_pkts Packets received successfully.
uint32_t	tx_pkts Packets transmitted successfully.
uint32_t	tx_err

	Transmit errors.

Detailed Description

Define the statistic and error counters for this IP instance.

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_scan_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_interface_hw_mode_t	hw_mode Hardware mode 802.11a/b/g/n.
uint8_t	rssi Signal Strength.
uint8_t	ssid [SF_WIFI_SSID_LENGTH+1] SSID name.
uint8_t	bssid [SF_WIFI_MAC_ADDR_LENGTH] Basic Service Set Identification (i.e. MAC address of Access Point)
uint8_t	channel Radio channel that the AP beacon was received on.
sf_wifi_security_type_t	security Security type.

sf_wifi_encryption_type_t	encryption Encryption type.
sf_wifi_bss_type_t	bss_type Network type.

Detailed Description

Define the structure to store the SSID scan information

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_wps_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_wps_mode_t	wps_mode WPS method Push-button or Pin.
uint8_t	wps_key [SF_WIFI_WPS_PIN_LENGTH+SF_WIFI_SIZE_FOR_NULL_BYTE]
uint8_t	timeout_seconds WPS timeout value in seconds.
void(*	p_callback)(sf_wifi_callback_args_t *p_args)

Detailed Description

Define the structure for WiFi WPS Control

Field Documentation

◆ p_callback

void(* sf_wifi_wps_t::p_callback) (sf_wifi_callback_args_t *p_args)
Pointer to callback function to be called on changed in client's connection status with AP or client connected/disconnected

◆ wps_key

uint8_t sf_wifi_wps_t::wps_key[SF_WIFI_WPS_PIN_LENGTH+SF_WIFI_SIZE_FOR_NULL_BYTE]
WPS pin. Used only with WPS pin method, PIN should be NULL terminated

---

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

void *	p_driver_handle

Detailed Description

WiFi Framework control structure

Field Documentation

◆ p_driver_handle

void* sf_wifi_ctrl_t::p_driver_handle
Storage for information needed for each WiFi device driver in the system.

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

ssp_err_t(*	open)(sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const *const p_cfg) Initializes the network interface for data transfers. More...
ssp_err_t(*	close)(sf_wifi_ctrl_t *const p_ctrl) De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	multicastListAdd)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr) Add the given MAC address to the multicast filter list. More...
ssp_err_t(*	multicastListDelete)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)
ssp_err_t(*	statisticsGet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_stats_t *const p_wifi_device_stats) Get the interface statistics. More...
ssp_err_t(*	transmit)(sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length) Transmit data packet. More...
ssp_err_t(*	provisioningSet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t const *const p_wifi_provisioning) Set WiFi module provisioning which will configure the module in AP or Client mode. More...

ssp_err_t(*	provisioningGet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t *const p_wifi_provisioning) Get the provisioning information of WiFi module. More...
ssp_err_t(*	infoGet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t *const p_wifi_info) Get WiFi module information. More...
ssp_err_t(*	scan)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t *const p_scan, uint8_t *const p_cnt) Scan for WiFi SSIDs. More...
ssp_err_t(*	ACLAdd)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Adds a MAC address to the Access Control List. Valid in AP mode only. More...
ssp_err_t(*	ACLDelete)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Deletes a MAC address from Access Control List. Valid in AP mode only. More...
ssp_err_t(*	macAddressGet)(sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_mac) Get WiFi module MAC address. More...
ssp_err_t(*	macAddressSet)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Set WiFi module MAC address. More...
ssp_err_t(*	wpsStart)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_wps_t const *const p_wps) Start WiFi WPS. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Framework API structure. Implementations will use the following API.

Field Documentation

◆ ACLAdd

ssp_err_t(* sf_wifi_api_t::ACLAdd) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)

Adds a MAC address to the Access Control List. Valid in AP mode only.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_mac	Pointer to MAC address

◆ ACLDelete

ssp_err_t(* sf_wifi_api_t::ACLDelete) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)

Deletes a MAC address from Access Control List. Valid in AP mode only.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_mac	Pointer to MAC address

◆ close

ssp_err_t(* sf_wifi_api_t::close) (sf_wifi_ctrl_t *const p_ctrl)

De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block for the WiFi module.

◆ infoGet

ssp_err_t(* sf_wifi_api_t::infoGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t *const p_wifi_info)

Get WiFi module information.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_wifi_info	Pointer to WiFi module information structure

◆ macAddressGet

ssp_err_t(* sf_wifi_api_t::macAddressGet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_mac)

Get WiFi module MAC address.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_mac	Pointer to MAC address

◆ macAddressSet

ssp_err_t(* sf_wifi_api_t::macAddressSet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)

Set WiFi module MAC address.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_mac	Pointer to MAC address

◆ multicastListAdd

ssp_err_t(* sf_wifi_api_t::multicastListAdd) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)

Add the given MAC address to the multicast filter list.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_mac_addr	Pointer to the Mac address.

◆ multicastListDelete

ssp_err_t(* sf_wifi_api_t::multicastListDelete) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)

Delete the given MAC address from the multicast filter list.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_mac_addr	Pointer to the Mac address.

◆ open

ssp_err_t(* sf_wifi_api_t::open) (sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const *const p_cfg)

Initializes the network interface for data transfers.

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to WiFi configuration structure.

◆ provisioningGet

ssp_err_t(* sf_wifi_api_t::provisioningGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t *const p_wifi_provisioning)

Get the provisioning information of WiFi module.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_wifi_provisioning	Pointer to WiFi provisioning structure

◆ provisioningSet

ssp_err_t(* sf_wifi_api_t::provisioningSet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t const *const p_wifi_provisioning)

Set WiFi module provisioning which will configure the module in AP or Client mode.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_wifi_provisioning	Pointer to WiFi provisioning structure

◆ scan

ssp_err_t(* sf_wifi_api_t::scan) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t *const p_scan, uint8_t *const p_cnt)

Scan for WiFi SSIDs.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_scan	Pointer to structure which will hold scan result. It is the responsibility of the caller to ensure that adequate space is available to hold scan results.
[in,out]	p_cnt	Pointer to variable, specifying maximum number of SSID's to scan and will be updated to number of actual SSIDs scanned by device

◆ statisticsGet

ssp_err_t(* sf_wifi_api_t::statisticsGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_stats_t *const p_wifi_device_stats)

Get the interface statistics.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_wifi_device_stats	Pointer to the WiFi module data statistics.

◆ transmit

ssp_err_t(* sf_wifi_api_t::transmit) (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length)

Transmit data packet.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_buf	Pointer to transmit buffer
[in]	length	Transmit buffer length

◆ versionGet

ssp_err_t(* sf_wifi_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	pointer to memory location to return version number

◆ wpsStart

ssp_err_t(* sf_wifi_api_t::wpsStart) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_wps_t const *const p_wps)

Start WiFi WPS.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_wps	Pointer to WiFi WPS configuration

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

sf_wifi_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI Framework Interface

#include <sf_wifi_api.h>

Data Fields

sf_wifi_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_wifi_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_wifi_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_wifi_api.h

5.1.2.49 SF WIFI NSAL Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF WIFI NSAL Framework Interface. More...

Data Structures

struct	sf_wifi_nsal_cfg_t
struct	sf_wifi_nsal_callback_args_t

Enumerations

enum	sf_wifi_nsal_zero_copy_t { SF_WIFI_NSAL_ZERO_COPY_DISABLE, SF_WIFI_NSAL_ZERO_COPY_ENABLE }

Detailed Description

RTOS-integrated SF WIFI NSAL Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI NSAL Framework.

Enumeration Type Documentation

◆ sf_wifi_nsal_zero_copy_t

enum sf_wifi_nsal_zero_copy_t	
Zero Copy Configuration Enumeration	
Enumerator	
SF_WIFI_NSAL_ZERO_COPY_DISABLE	Zero copy is disabled.
SF_WIFI_NSAL_ZERO_COPY_ENABLE	Zero copy is enabled.

sf_wifi_nsal_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI NSAL Interface

#include <sf_wifi_nsal_api.h>

Data Fields

sf_wifi_nsal_zero_copy_t	tx_zero_copy Transmit path zero copy support.
sf_wifi_nsal_zero_copy_t	rx_zero_copy Receive path zero copy support.
uint8_t *	p_tx_packet_buffer Pointer to Tx buffer used to consolidate data from chained NetX packets.

Detailed Description

Define the NSAL configuration parameters

The documentation for this struct was generated from the following file:

- sf_wifi_nsal_api.h

sf_wifi_nsal_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI NSAL Interface

#include <sf_wifi_nsal_api.h>

Data Fields

NX_INTERFACE *	p_interface Pointer to NetX interface.
NX_IP *	p_ip Pointer to NetX IP.
sf_wifi_nsal_cfg_t *	p_wifi_nsal_cfg pointer to NSAL configuration

Detailed Description

Define the NSAL callback arguments

The documentation for this struct was generated from the following file:

- sf_wifi_nsal_api.h

5.1.2.50 SF WIFI On-Chip Stack Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF WIFI On-Chip Stack Interface. More...

Data Structures

struct	sf_wifi_onchip_stack_ip_cfg_t
struct	sf_wifi_onchip_stack_cfg_t
struct	sf_wifi_onchip_stack_ctrl_t
struct	sf_wifi_onchip_stack_api_t
struct	sf_wifi_onchip_stack_instance_t

Macros

#define	SF_WIFI_ONCHIP_STACK_API_VER_MAJOR (2U)
#define	SF_WIFI_ONCHIP_STACK_API_VER_MINOR (0U)

Enumerations

enum	sf_wifi_onchip_stack_ip_addr_mode_t { SF_WIFI_IP_ADDR_GET, SF_WIFI_IP_ADDR_STATIC, SF_WIFI_IP_ADDR_DHCP }

Detailed Description

RTOS-integrated SF WIFI On-Chip Stack Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI Framework.

Macro Definition Documentation

◆ SF_WIFI_ONCHIP_STACK_API_VER_MAJOR

#define SF_WIFI_ONCHIP_STACK_API_VER_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_WIFI_ONCHIP_STACK_API_VER_MINOR

#define SF_WIFI_ONCHIP_STACK_API_VER_MINOR (0U)
Minor Version of the API defined in this file

Enumeration Type Documentation

◆ sf_wifi_onchip_stack_ip_addr_mode_t

enum sf_wifi_onchip_stack_ip_addr_mode_t	
IP addressing modes	
Enumerator	
SF_WIFI_IP_ADDR_GET	Read the IP address assigned to interface.
SF_WIFI_IP_ADDR_STATIC	Statically configure the IP address.
SF_WIFI_IP_ADDR_DHCP	Get the IP address from DHCP server, dynamic assignment.

sf_wifi_onchip_stack_ip_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI On-Chip Stack Interface

#include <sf_wifi_onchip_stack_api.h>

Data Fields

sf_wifi_onchip_stack_ip_addr_mode_t	ip_addr_mode Addressing mode.
sf_wifi_ip_addr_t	ip_addr Interface IP address.
sf_wifi_ip_addr_t	netmask Interface netmask.
sf_wifi_ip_addr_t	gateway Interface Gateway.

Detailed Description

Define IP Interface configuration information

The documentation for this struct was generated from the following file:

- sf_wifi_onchip_stack_api.h

sf_wifi_onchip_stack_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI On-Chip Stack Interface

#include <sf_wifi_onchip_stack_api.h>

Data Fields

sf_wifi_instance_t const *	p_lower_lvl_wifi Pointer to SF WiFi instance.
void *	p_extend Extended configuration.

Detailed Description

Define the WiFi configuration parameters

The documentation for this struct was generated from the following file:

- sf_wifi_onchip_stack_api.h

sf_wifi_onchip_stack_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI On-Chip Stack Interface

#include <sf_wifi_onchip_stack_api.h>

Data Fields

sf_wifi_instance_t *	p_lower_lvl_wifi Pointer to SF WiFi instance. More...

Detailed Description

WiFi Framework control structure

Field Documentation

◆ p_lower_lvl_wifi

sf_wifi_instance_t* sf_wifi_onchip_stack_ctrl_t::p_lower_lvl_wifi
Pointer to SF WiFi instance. Storage for information needed for each WiFi device driver in the system.

The documentation for this struct was generated from the following file:

- sf_wifi_onchip_stack_api.h

sf_wifi_onchip_stack_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI On-Chip Stack Interface

#include <sf_wifi_onchip_stack_api.h>

Data Fields

ssp_err_t(*	open)(sf_wifi_onchip_stack_ctrl_t *p_ctrl, sf_wifi_onchip_stack_cfg_t const *const p_cfg) Pointer to function which initializes the network interface for data transfers. More...
ssp_err_t(*	close)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl) Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	ipAddressCfg)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_onchip_stack_ip_cfg_t *const p_cfg) Configures IP address of the interface. More...
ssp_err_t(*	dhcpServerStart)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl,

	sf_wifi_ip_addr_t const *const p_start_ip, sf_wifi_ip_addr_t const *const p_end_ip) Starts DHCP server on the interface. More...
ssp_err_t(*	dhcpServerStop)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl) Stop DHCP server on the interface. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Framework API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_wifi_onchip_stack_api_t::close) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl)

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block

◆ dhcpServerStart

ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStart) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_ip_addr_t const *const p_start_ip, sf_wifi_ip_addr_t const *const p_end_ip)

Starts DHCP server on the interface.

Parameters

[in]	p_ctrl	Pointer to the control block
[in]	p_start_ip	Pointer to start IP address
[in]	p_end_ip	Pointer to end IP address

◆ dhcpServerStop

ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStop) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl)

Stop DHCP server on the interface.

Parameters

[in]	p_ctrl	Pointer to the control block

◆ ipAddressCfg

ssp_err_t(* sf_wifi_onchip_stack_api_t::ipAddressCfg) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_onchip_stack_ip_cfg_t *const p_cfg)

Configures IP address of the interface.

Parameters

[in]	p_ctrl	Pointer to the control block
[in,out]	p_cfg	Pointer to IP configuration structure.

◆ open

ssp_err_t(* sf_wifi_onchip_stack_api_t::open) (sf_wifi_onchip_stack_ctrl_t *p_ctrl, sf_wifi_onchip_stack_cfg_t const *const p_cfg)

Pointer to function which initializes the network interface for data transfers.

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to configuration structure.

◆ versionGet

ssp_err_t(* sf_wifi_onchip_stack_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters**		
[out]	p_version	pointer to memory location to return version number

---

The documentation for this struct was generated from the following file:

- sf_wifi_onchip_stack_api.h

sf_wifi_onchip_stack_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI On-Chip Stack Interface

#include <sf_wifi_onchip_stack_api.h>

Data Fields

sf_wifi_onchip_stack_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_wifi_onchip_stack_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_wifi_onchip_stack_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

SF WiFi On Chip Stack Instance structure

The documentation for this struct was generated from the following file:

- sf_wifi_onchip_stack_api.h

5.1.2.51 SF WIFI QCA4010 Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF WIFI QCA4010 Framework Interface. More...

Data Structures

struct	sf_wifi_qca4010_status_t
struct	sf_wifi_qca4010_at_cmd_set_t
struct	sf_wifi_qca4010_cmd_resp_t
struct	sf_wifi_qca4010_cfg_t
struct	sf_wifi_qca4010_provisioning_t
struct	sf_wifi_qca4010_scan_t
struct	sf_wifi_qca4010_uart_extend_cfg_t
struct	sf_wifi_qca4010_queue_cfg_t
struct	sf_wifi_qca4010_extended_cfg_t
struct	sf_wifi_qca4010_ctrl_t
struct	sf_wifi_qca4010_api_t
struct	sf_wifi_qca4010_instance_t

Macros

#define	SF_WIFI_QCA4010_API_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_API_VERSION_MINOR (0U)
#define	SF_WIFI_QCA4010_SSID_LENGTH (32U) WiFi SSID length.

#define	SF_WIFI_QCA4010_BSSID_LENGTH (6U) WiFi BSSID length.
#define	SF_WIFI_QCA4010_SECURITY_KEY_LENGTH (128U) WiFi Security Key length.
#define	SF_WIFI_QCA4010_MUTEX_GET_TIMEOUT_TICKS (200U) Default timeout for getting mutex.
#define	MAC_ADDRESS_LEN (18U)
#define	PHY_MODE_LEN (6U)
#define	ACCESS_MODE_LEN (10U)
#define	POWER_MODE_LEN (10U)

Enumerations

enum	sf_wifi_qca4010_at_cmd_index_t { SF_WIFI_QCA4010_AT_CMD_INDEX_AT = 0, SF_WIFI_QCA4010_AT_CMD_INDEX_ATZ0, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BAUD_CHECK, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BAUD_SET, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SAVE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ECHO, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SET_HARDWARE_MODE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_RESULT_CODE_FROMAT, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_UART_TRANSMISSION_FLOW_CONTROL_QUERY, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_UART_TRANSMISSION_FLOW_CONTROL, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ESCAPE_GUARD_TIME, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BUFFER_SIZE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_DISCONNECT_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BUFFER_TIMEOUT, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SCAN_SSID, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_AP_MODE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_OPEN_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_WPA_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ASSIGN_AP_CHANNEL, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_OPEN_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_WPA_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ASSIGN_WEP_KEY, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_WEP_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_WEP_AP, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_WIFI_STATUS, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_DISCONNECT_ACCESS_POINT,

	SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INIT_SECOND_UART, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CHANGE_UART_ASSIGNMENT, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CHANGE_UART_ASSIGNMENT_IN_CLOSE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SWITCH_BACK_TO_DATA_MODE, SF_WIFI_QCA4010_AT_CMD_INDEX_AT_EXIT_DATA_MODE }
enum	sf_wifi_qca4010_interface_hw_mode_t { SF_WIFI_QCA4010_INTERFACE_HW_MODE_11B, SF_WIFI_QCA4010_INTERFACE_HW_MODE_11G, SF_WIFI_QCA4010_INTERFACE_HW_MODE_11N }
enum	sf_wifi_qca4010_reset_type_t { SF_WIFI_QCA4010_RESET_TYPE_SOFT, SF_WIFI_QCA4010_RESET_TYPE_HARD }
enum	sf_wifi_qca4010_interface_mode_t { SF_WIFI_QCA4010_INTERFACE_MODE_AP, SF_WIFI_QCA4010_INTERFACE_MODE_CLIENT }
enum	sf_wifi_qca4010_security_type_t { SF_WIFI_QCA4010_SECURITY_TYPE_OPEN = 0, SF_WIFI_QCA4010_SECURITY_TYPE_WEP, SF_WIFI_QCA4010_SECURITY_TYPE_WPA, SF_WIFI_QCA4010_SECURITY_TYPE_WPA2 }
enum	sf_wifi_qca4010_encryption_type_t { SF_WIFI_QCA4010_ENCRYPTION_TYPE_TKIP = 0, SF_WIFI_QCA4010_ENCRYPTION_TYPE_CCMP }

Detailed Description

RTOS-integrated SF WIFI QCA4010 Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI QCA4010 Framework.

Macro Definition Documentation

◆ ACCESS_MODE_LEN

#define ACCESS_MODE_LEN (10U)
Mode len

◆ MAC_ADDRESS_LEN

#define MAC_ADDRESS_LEN (18U)
Mac address len

◆ PHY_MODE_LEN

#define PHY_MODE_LEN (6U)
phy mode len

◆ POWER_MODE_LEN

#define POWER_MODE_LEN (10U)
power mode len

◆ SF_WIFI_QCA4010_API_VERSION_MAJOR

#define SF_WIFI_QCA4010_API_VERSION_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_WIFI_QCA4010_API_VERSION_MINOR

#define SF_WIFI_QCA4010_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

Enumeration Type Documentation

◆ sf_wifi_qca4010_at_cmd_index_t

enum sf_wifi_qca4010_at_cmd_index_t	
Enumeration for AT command index	
Enumerator	
SF_WIFI_QCA4010_AT_CMD_INDEX_AT	Index for Command AT.
SF_WIFI_QCA4010_AT_CMD_INDEX_ATZ0	Index for Command ATZ0.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BAUD_CHECK	Index for Command to check whether modem is responding after baud update.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BAUD_SET	Index for Command to set baud rate of wifi.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SAVE	Index for Command AT&W.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ECHO	Index for Command ATE0.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SET_HARDWARE_MODE	Index for Command ATWPHYMODE.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_RESULT_CODE_FROMAT	Index for Command ATV.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_UART_TRANSMISSION_FLOW_CONTROL_QUERY	Index for Command ATS108?
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_UART_TRANSMISSION_FLOW_CONTROL	Index for Command ATS108=1.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ESCAPE_GUARD_TIME	Index for Command ATS12=1.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BUFFER_SIZE	Index for Command Set buffer size for socket receive (ATBSIZE)
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_DISCONNECT_AP	Index for Command disconnect from currently connected Access Point.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_BUFFER_TIMEOUT	Index for Command to the system wait to send UART-received data to network.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SCAN_SSID	Index for Command to the scan available ssid.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_AP_MODE	Index for Command to initiate AP mode.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_OPEN_AP	Index for Command to the connect open AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_WPA_AP	Index for Command to the connect to WPA-

	configured Access Point.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ASSIGN_AP_CHANNEL	Index for Command to assign channel to AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_OPEN_AP	Index for Command to initiate Open AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_WPA_AP	Index for Command to initiate WPA/WPA2 AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_ASSIGN_WEP_KEY	Index for Command to assign WEP key.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INITIATE_WEP_AP	Index for Command to initiate WEP AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CONNECT_WEP_AP	Index for Command to connect to WEP AP.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_WIFI_STATUS	Index for Command to the get wifi status.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_DISCONNECT_ACCESS_POINT	Index for Command to disconnect current access point.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_INIT_SECOND_UART	Index for Command to init second uart.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CHANGE_UART_ASSIGNMENT	Change UART assignments for command and data.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_CHANGE_UART_ASSIGNMENT_IN_CLOSE	Change UART assignments for command and data in close if num_uart = 2.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_SWITCH_BACK_TO_DATA_MODE	Index for Command to switch to data mode.
SF_WIFI_QCA4010_AT_CMD_INDEX_AT_EXIT_DATA_MODE	Index for Command to switch to command mode.

◆ sf_wifi_qca4010_encryption_type_t

enum sf_wifi_qca4010_encryption_type_t	
WiFi Encryption type	
Enumerator	
SF_WIFI_QCA4010_ENCRYPTION_TYPE_TKIP	Temporal Key Integrity Protocol.
SF_WIFI_QCA4010_ENCRYPTION_TYPE_CCMP	CTR mode with CBC-MAC Protocol.

◆ sf_wifi_qca4010_interface_hw_mode_t

enum sf_wifi_qca4010_interface_hw_mode_t	
WiFi Hardware mode	
Enumerator	
SF_WIFI_QCA4010_INTERFACE_HW_MODE_11B	802.11b
SF_WIFI_QCA4010_INTERFACE_HW_MODE_11G	802.11g
SF_WIFI_QCA4010_INTERFACE_HW_MODE_11N	802.11n

◆ sf_wifi_qca4010_interface_mode_t

enum sf_wifi_qca4010_interface_mode_t	
WiFi Interface mode	
Enumerator	
SF_WIFI_QCA4010_INTERFACE_MODE_AP	Access Point mode.
SF_WIFI_QCA4010_INTERFACE_MODE_CLIENT	Station Mode.

◆ sf_wifi_qca4010_reset_type_t

enum sf_wifi_qca4010_reset_type_t	
Wifi Module reset type	
Enumerator	
SF_WIFI_QCA4010_RESET_TYPE_SOFT	Soft reset module using AT command.
SF_WIFI_QCA4010_RESET_TYPE_HARD	Hard reset module by toggling Reset Pin.

◆ sf_wifi_qca4010_security_type_t

enum sf_wifi_qca4010_security_type_t	
WiFi Security type	
Enumerator	
SF_WIFI_QCA4010_SECURITY_TYPE_OPEN	Open. No encryption used.
SF_WIFI_QCA4010_SECURITY_TYPE_WEP	10 or 26 digit hexadecimal string
SF_WIFI_QCA4010_SECURITY_TYPE_WPA	WiFi Protected Access.
SF_WIFI_QCA4010_SECURITY_TYPE_WPA2	WiFi Protected Access v2.

sf_wifi_qca4010_status_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uint8_t	ssid [SF_WIFI_QCA4010_SSID_LENGTH] SSID.
uint8_t	phy_mode [PHY_MODE_LEN] Phy mode.
uint8_t	mac_addr [MAC_ADDRESS_LEN] Mac addr.
uint8_t	mode [ACCESS_MODE_LEN] Mode.
uint8_t	channel [MAX_CHANNEL_LENGTH] Channel.

Detailed Description

Network Status information

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_at_cmd_set_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uint8_t *	p_cmd AT Command.
uint8_t *	p_success_resp Success response string.
uint16_t	max_resp_length Maximum length of expected response.
uint32_t	resp_wait_time AT command response wait time in milliseconds.
uint16_t	retry_delay Delay between AT command retry.

Detailed Description

Structure defining AT commands parameters

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_cmd_resp_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uint8_t *	p_buff
uint32_t	buff_len

Detailed Description

Modem Command/Response structure used to send Custom AT command and to receive response for the same

Field Documentation

◆ buff_len

uint32_t sf_wifi_qca4010_cmd_resp_t::buff_len

AT command/Response buffer length. In case of Response this is both in and out parameter. Input is the length of buffer pointed by p_buff and output is number of bytes of response copied by framework. In case of command it is length of command in p_buff

◆ p_buff

uint8_t* sf_wifi_qca4010_cmd_resp_t::p_buff

AT command/Response buffer. In case of AT command it is input buffer in which user should pass custom command to be sent. In case of Response it is output buffer in which framework will fill the response

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

sf_wifi_qca4010_interface_hw_mode_t	hw_mode Modulation type: 11b/g/n.
const uart_instance_t *	p_uart_instances [SF_WIFI_QCA4010_CFG_MAX_NUMBER_UART_PORTS] SCI UART instances.
const uint32_t	num_uarts Number of UART interfaces to use.
void const *	p_context User defined context passed into callback function.
void const *	p_extend Instance specific configuration.
sf_wifi_qca4010_at_cmd_set_t const *	p_cmd_set Instance specific command set.

Detailed Description

Define the WiFi configuration parameters

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_provisioning_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

sf_wifi_qca4010_interface_mode_t	mode Select AP or Client mode.
uint8_t	channel RF Channel number.
uint8_t	ssid [SF_WIFI_QCA4010_SSID_LENGTH+1] SSID.
sf_wifi_qca4010_security_type_t	security Security type.
sf_wifi_qca4010_encryption_type_t	encryption Encryption type.
uint8_t	key [SF_WIFI_QCA4010_SECURITY_KEY_LENGTH] Pre-shared key.
uint8_t	index WEP Key index, value ranges between 1 to 4.

Detailed Description

WiFi Provisioning parameters

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_scan_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uint8_t	ssid [SF_WIFI_QCA4010_SSID_LENGTH] SSID name.
uint8_t	bssid [MAC_ADDRESS_LEN] Basic Service Set Identification (i.e. MAC address of Access Point)
uint8_t	channel [MAX_CHANNEL_LENGTH] Radio channel that the AP beacon was received on.
uint8_t	security Security type.

Detailed Description

Define the structure to store the SSID scan information

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_uart_extend_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uart_instance_t const *	p_r_uart_instance Lower level HAL driver instance.

Detailed Description

UART configuration

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_queue_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

uint32_t *	p_queue_buffer Queue buffer.
uint32_t	queue_size Queue Size.
uint8_t	ok_check_index Variable to store index for data checking success string response.
uint8_t	error_check_index Variable to store index for data checking error string response.

TX_QUEUE *	p_cmd_queue_ptr Queue.

Detailed Description

Queue configuration

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

sf_wifi_qca4010_queue_cfg_t *	p_queue_cfg Queue configuration.
TX_EVENT_FLAGS_GROUP *	p_eventflag [2] Pointer to the event flag object for UART data transfer.
ioport_port_pin_t	pin_reset Port pin used for resetting wifi module.
ioport_level_t	reset_level Module reset level.
void *	p_module_extended_cfg Instance specific module configuration.

Detailed Description

SF wifi framework extended configuration

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

void *	p_driver_handle

Detailed Description

WiFi Framework control structure

Field Documentation

◆ p_driver_handle

void* sf_wifi_qca4010_ctrl_t::p_driver_handle
Storage for information needed for each WiFi device driver in the system.

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

ssp_err_t(*	open)(sf_wifi_qca4010_ctrl_t *p_ctrl, sf_wifi_qca4010_cfg_t const *const p_cfg) Initializes the network interface for data transfers. More...
ssp_err_t(*	close)(sf_wifi_qca4010_ctrl_t *const p_ctrl) De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt. More...
ssp_err_t(*	provisioningSet)(sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_provisioning_t const *const p_wifi_provisioning) Set WiFi module provisioning which will configure the module in AP or Client mode. More...
ssp_err_t(*	wifiStatusGet)(sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_status_t *const p_wifi_status) Get WiFi module information. More...
ssp_err_t(*	scan)(sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_scan_t *const p_scan, uint8_t count) Scan for WiFi SSIDs. More...
ssp_err_t(*	CommandSend)(sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_cmd_resp_t *const p_input_at_command, sf_wifi_qca4010_cmd_resp_t *const p_output, uint32_t const timeout) Send AT command given by user. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Framework API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_wifi_qca4010_api_t::close) (sf_wifi_qca4010_ctrl_t *const p_ctrl)

De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block for the WiFi module.

◆ CommandSend

ssp_err_t(* sf_wifi_qca4010_api_t::CommandSend) (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_cmd_resp_t *const p_input_at_command, sf_wifi_qca4010_cmd_resp_t *const p_output, uint32_t const timeout)

Send AT command given by user.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_input_at_command	Pointer to structure which contains Modem command to send
[in,out]	p_output	Pointer to buffer in which response will be sent to user, Also user will pass the size of the buffer which is pointed by p_output
[in]	timeout	Timeout for which framework will wait for response in milliseconds

◆ open

ssp_err_t(* sf_wifi_qca4010_api_t::open) (sf_wifi_qca4010_ctrl_t *p_ctrl, sf_wifi_qca4010_cfg_t const *const p_cfg)

Initializes the network interface for data transfers.

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to WiFi configuration structure.

◆ provisioningSet

ssp_err_t(* sf_wifi_qca4010_api_t::provisioningSet) (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_provisioning_t const *const p_wifi_provisioning)

Set WiFi module provisioning which will configure the module in AP or Client mode.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[in]	p_wifi_provisioning	Pointer to WiFi provisioning structure

◆ scan

ssp_err_t(* sf_wifi_qca4010_api_t::scan) (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_scan_t *const p_scan, uint8_t count)

Scan for WiFi SSIDs.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_scan	Pointer to structure which will hold scan result. It is the responsibility of the caller to ensure that adequate space is available to hold scan results.
[in,out]	count	Variable specifying maximum number of SSID's to scan and will be updated to number of actual SSIDs scanned by device

◆ versionGet

ssp_err_t(* sf_wifi_qca4010_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	pointer to memory location to return version number

◆ wifiStatusGet

ssp_err_t(* sf_wifi_qca4010_api_t::wifiStatusGet) (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_status_t *const p_wifi_status)

Get WiFi module information.

Parameters

[in]	p_ctrl	Pointer to the control block for the WiFi module.
[out]	p_wifi_info	Pointer to WiFi module information structure

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

sf_wifi_qca4010_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 Framework Interface

#include <sf_wifi_qca4010_api.h>

Data Fields

sf_wifi_qca4010_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_wifi_qca4010_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_wifi_qca4010_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_api.h

5.1.2.52 SF WIFI QCA4010 On-Chip Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF Socket Wifi Framework Interface. More...

Data Structures

struct	sf_wifi_qca4010_ip_addr_t
struct	sf_wifi_qca4010_onchip_stack_ip_cfg_t
struct	sf_wifi_qca4010_onchip_stack_ctrl_t
struct	sf_wifi_qca4010_onchip_stack_cfg_t
struct	sf_wifi_qca4010_onchip_stack_api_t
struct	sf_wifi_qca4010_onchip_stack_instance_t

Macros

#define	SF_WIFI_QCA4010_ONCHIP_API_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_ONCHIP_API_VERSION_MINOR (0U)
#define	IP_ADDRESS(a, b, c, d) ((((ULONG)a) << 24) | (((ULONG)b) << 16) | (((ULONG)c) << 8) | ((ULONG)d))

Enumerations

enum	sf_wifi_qca4010_onchip_at_cmd_index_t { SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_ENABLE_DHCP = 0, SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_GET_IP_ADDRESS, SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_STATIC_IP_ADDRESS, SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_PING_IP_ADDRESS, SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_START_DHCP_SERVER, SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_STOP_DHCP_SERVER }
enum	sf_wifi_qca4010_onchip_stack_ip_addr_mode_t { SF_WIFI_QCA4010_IP_ADDR_MODE_STATIC, SF_WIFI_QCA4010_IP_ADDR_MODE_DHCP }
enum	sf_wifi_qca4010_ip_addr_version_t { SF_WIFI_QCA4010_IP_ADDR_VERSION_4, SF_WIFI_QCA4010_IP_ADDR_VERSION_6 }

Detailed Description

RTOS-integrated SF Socket Wifi Framework Interface.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI QCA4010 Framework.

Macro Definition Documentation

◆ IP_ADDRESS

#define IP_ADDRESS (a, b, c, d) ((((ULONG)a) << 24) | (((ULONG)b) << 16) | (((ULONG)c) << 8) | ((ULONG)d))
IP Address Generation Macro

◆ SF_WIFI_QCA4010_ONCHIP_API_VERSION_MAJOR

#define SF_WIFI_QCA4010_ONCHIP_API_VERSION_MAJOR (2U)
SF wifi onchip APIs Major Version

◆ SF_WIFI_QCA4010_ONCHIP_API_VERSION_MINOR

#define SF_WIFI_QCA4010_ONCHIP_API_VERSION_MINOR (0U)
SF wifi onchip APIs Minor Version

Enumeration Type Documentation

◆ sf_wifi_qca4010_ip_addr_version_t

enum sf_wifi_qca4010_ip_addr_version_t	
IP address version	
Enumerator	
SF_WIFI_QCA4010_IP_ADDR_VERSION_4	IPv4 address.
SF_WIFI_QCA4010_IP_ADDR_VERSION_6	IPv6 address.

◆ sf_wifi_qca4010_onchip_at_cmd_index_t

enum sf_wifi_qca4010_onchip_at_cmd_index_t	
Enumeration for AT command index	
Enumerator	
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_ENABLE_DHCP	Index for Command to enable DHCP.
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_GET_IP_ADDRESS	Index for Command to IP address get.
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_STATIC_IP_ADDRESS	Index for command to static ip address get.
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_PING_IP_ADDRESS	Index for command to static ip address get.
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_START_DHCP_SERVER	Index for Command to start DHCP server.
SF_WIFI_QCA4010_ONCHIP_AT_CMD_INDEX_STOP_DHCP_SERVER	Index for Command to stop DHCP server.

◆ sf_wifi_qca4010_onchip_stack_ip_addr_mode_t

enum sf_wifi_qca4010_onchip_stack_ip_addr_mode_t	
IP addressing modes	
Enumerator	
SF_WIFI_QCA4010_IP_ADDR_MODE_STATIC	Statically configure the IP address.
SF_WIFI_QCA4010_IP_ADDR_MODE_DHCP	Get the IP address from DHCP server, dynamic assignment.

sf_wifi_qca4010_ip_addr_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

sf_wifi_qca4010_ip_addr_version_t	version

	IP Address Version : v4 or v6.
union {	
}	addr IP address.

Detailed Description

IP address information

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

sf_wifi_qca4010_onchip_stack_ip_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

sf_wifi_qca4010_onchip_stack_ip_addr_mode_t	ip_addr_mode Addressing mode.
sf_wifi_qca4010_ip_addr_t	ip_addr Interface IP address.
sf_wifi_qca4010_ip_addr_t	netmask Interface netmask.
sf_wifi_qca4010_ip_addr_t	gateway Interface Gateway.

Detailed Description

Define IP Interface configuration information

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

sf_wifi_qca4010_onchip_stack_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

sf_wifi_qca4010_instance_t *	p_lower_lvl_wifi_qca4010 Pointer to SF on-chip stack instance. More...

Detailed Description

Socket Interface control structure

Field Documentation

◆ p_lower_lvl_wifi_qca4010

sf_wifi_qca4010_instance_t* sf_wifi_qca4010_onchip_stack_ctrl_t::p_lower_lvl_wifi_qca4010

Pointer to SF on-chip stack instance.

Storage for information needed for each WiFi device driver in the system.

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

sf_wifi_qca4010_onchip_stack_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip

Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

sf_wifi_qca4010_instance_t *	p_lower_lvl_wifi_qca4010 Pointer to SF on-chip stack instance.

Detailed Description

Socket Interface configuration structure

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

sf_wifi_qca4010_onchip_stack_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

ssp_err_t(*	open)(sf_wifi_qca4010_onchip_stack_ctrl_t *p_ctrl, sf_wifi_qca4010_onchip_stack_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl)
ssp_err_t(*	ipAddressCfg)(sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_qca4010_onchip_stack_ip_cfg_t *const p_ip_cfg)
ssp_err_t(*	ping)(sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_ip_addr, uint32_t count, uint32_t interval_ms)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	dhcpServerStart)(sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip)

ssp_err_t(*	dhcpServerStop)(sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip)

Detailed Description

Framework API structure. Implementations will use the following API.

Field Documentation

◆ close

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::close) (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl)

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Parameters

[in,out]	p_ctrl	Pointer to the control block

◆ dhcpServerStart

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::dhcpServerStart) (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip)

Starts DHCP server

Parameters

[in]	p_ctrl	Pointer to the control block
[in]	p_start_ip	Pointer to Start IP address
[in]	p_end_ip	Pointer to End IP address

◆ dhcpServerStop

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::dhcpServerStop) (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip)

Stops DHCP server

Parameters

[in]	p_ctrl	Pointer to the control block
[in]	p_start_ip	Pointer to Start IP address
[in]	p_end_ip	Pointer to End IP address

◆ ipAddressCfg

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::ipAddressCfg) (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_qca4010_onchip_stack_ip_cfg_t *const p_ip_cfg)

Configures IP address of the interface.

Parameters

[in]	p_ctrl	Pointer to the control block
[in,out]	p_ip_cfg	Pointer to IP configuration structure.

◆ open

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::open) (sf_wifi_qca4010_onchip_stack_ctrl_t *p_ctrl, sf_wifi_qca4010_onchip_stack_cfg_t const *const p_cfg)

Pointer to function which initializes the network interface for data transfers

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to configuration structure.

◆ ping

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::ping) (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_ip_addr, uint32_t count, uint32_t interval_ms)

Configures IP address of the interface.

Parameters

[in]	p_ctrl	Pointer to the control block
[in]	p_ip_addr	Pointer to IP address to ping
[in]	count	Number of ping attempts
[in]	interval_ms	Timeout in milliseconds

◆ versionGet

ssp_err_t(* sf_wifi_qca4010_onchip_stack_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters**		
[out]	p_version	pointer to memory location to return version number

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

sf_wifi_qca4010_onchip_stack_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF WIFI QCA4010 On-Chip Interface

#include <sf_wifi_qca4010_onchip_stack_api.h>

Data Fields

sf_wifi_qca4010_onchip_stack_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_wifi_qca4010_onchip_stack_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_wifi_qca4010_onchip_stack_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

SF WiFi On Chip Stack Instance structure

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_onchip_stack_api.h

5.1.2.53 SF Socket WIFI Framework Interface

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated SF Socket WIFI Framework Interface. More...

Data Structures

struct	in_addr
struct	sockaddr
struct	sockaddr_in
struct	ulpgn_socket_t
struct	sf_wifi_qca4010_socket_ctrl_t
struct	sf_wifi_qca4010_socket_cfg_t
struct	sf_wifi_qca4010_socket_api_t
struct	sf_wifi_qca4010_socket_instance_t

Macros

#define	SF_WIFI_QCA4010_SOCKET_API_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_SOCKET_API_VERSION_MINOR (0U)

Typedefs

typedef int32_t	socklen_t

Enumerations

enum	sf_wifi_qca4010_socket_at_cmd_index_t { SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SOCKET_INDEX = 0, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SOCKET_CREATE, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_TCP_SERVER, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_TCP_CLIENT, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_SERVER, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_CLIENT, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_EXIT_TRANSPARENT_MODE,

	SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_CLOSE_NETWORK_SOCKET, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_DATA_SEND, SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SWITCH_TO_DATA_MODE }

enum	sf_wifi_socket_type_t {
	SOCK_STREAM = 0, SOCK_DGRAM, SOCK_STREAM = 1,
	SOCK_DGRAM,
	SOCK_RAW
	}

enum	sf_wifi_qca4010_socket_type_t {
	SF_WIFI_QCA4010_SOCKET_TYPE_INVALID,
	SF_WIFI_QCA4010_SOCKET_TYPE_TCP_SERVER,
	SF_WIFI_QCA4010_SOCKET_TYPE_UDP_SERVER,
	SF_WIFI_QCA4010_SOCKET_TYPE_TCP_CLIENT,
	SF_WIFI_QCA4010_SOCKET_TYPE_UDP_CLIENT
	}

Detailed Description

RTOS-integrated SF Socket WIFI Framework Interface.

Summary

This SSP Interface provides access OnChip stack Socket API.

Macro Definition Documentation

◆ SF_WIFI_QCA4010_SOCKET_API_VERSION_MAJOR

#define SF_WIFI_QCA4010_SOCKET_API_VERSION_MAJOR (2U)
Major Version of the API defined in this file

◆ SF_WIFI_QCA4010_SOCKET_API_VERSION_MINOR

#define SF_WIFI_QCA4010_SOCKET_API_VERSION_MINOR (0U)
Minor Version of the API defined in this file

Typedef Documentation

◆ socklen_t

typedef int32_t socklen_t
Socket address Length

Enumeration Type Documentation

◆ sf_wifi_qca4010_socket_at_cmd_index_t

enum sf_wifi_qca4010_socket_at_cmd_index_t	
Socket specific AT command enumeration	
Enumerator	
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SOCKET_INDEX	Socket index.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SOCKET_CREATE	Socket create.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_TCP_SERVER	TCP server.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_TCP_CLIENT	TCP client.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_SERVER	UDP server.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_CLIENT	UDP client.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_EXIT_TRANSPARENT_MODE	Exit from transparent mode.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_CLOSE_NETWORK_SOCKET	Close network socket.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_UDP_DATA_SEND	Send data to UDP client.
SF_WIFI_QCA4010_SOCKET_AT_CMD_INDEX_SWITCH_TO_DATA_MODE	Switch to data mode.

◆ sf_wifi_qca4010_socket_type_t

enum sf_wifi_qca4010_socket_type_t	
Enumeration for socket type	
Enumerator	
SF_WIFI_QCA4010_SOCKET_TYPE_INVALID	Invalid socket type.
SF_WIFI_QCA4010_SOCKET_TYPE_TCP_SERVER	Data stream socket.
SF_WIFI_QCA4010_SOCKET_TYPE_UDP_SERVER	Datagram socket.
SF_WIFI_QCA4010_SOCKET_TYPE_TCP_CLIENT	Data stream socket.
SF_WIFI_QCA4010_SOCKET_TYPE_UDP_CLIENT	Datagram socket.

◆ sf_wifi_socket_type_t

enum sf_wifi_socket_type_t	
Type of Socket	
Enumerator	
SOCK_STREAM	TCP Socket.
SOCK_DGRAM	UDP Socket.
SOCK_STREAM	TCP Socket.
SOCK_DGRAM	UDP Socket.
SOCK_RAW	RAW Socket.

ulpgn_socket_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

uint32_t	socket_create_flag

	Current socket status.
uint8_t	socket_recv_buff [SF_WIFI_QCA4010_SOCKET_RECEIVE_BUFFER_SIZE] Number of queue handler. More...

Detailed Description

Silex ULPGN Wifi internal socket instance structure

Field Documentation

◆ socket_recv_buff

uint8_t ulpgn_socket_t::socket_recv_buff[SF_WIFI_QCA4010_SOCKET_RECEIVE_BUFFER_SIZE]
Number of queue handler. Socket receive buffer used by byte queue

---

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_socket_api.h

sf_wifi_qca4010_socket_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

ulpgn_socket_t	sockets [NUMBER_OF_SOCKET_INSTANCES] Internal socket instances.
sf_wifi_qca4010_onchip_stack_instance_t *	p_lower_lvl_onchip_wifi_qca4010 low level wifi interface

Detailed Description

Socket Interface control structure

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_socket_api.h

sf_wifi_qca4010_socket_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

sf_wifi_qca4010_onchip_stack_instance_t *	p_lower_lvl_onchip_wifi_qca4010 Pointer to SF on-chip stack instance.
void *	p_extend Extended configuration.

Detailed Description

Socket Interface configuration structure

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_socket_api.h

sf_wifi_qca4010_socket_api_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

ssp_err_t(*	open)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, sf_wifi_qca4010_socket_cfg_t const *const p_cfg) Open lower level driver. More...
ssp_err_t(*	close)(sf_wifi_qca4010_socket_ctrl_t *const p_ctrl) Close lower level driver. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...
ssp_err_t(*	socketCreate)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, sf_wifi_socket_type_t type, uint8_t ipversion) Create a socket. More...
ssp_err_t(*	socketConnect)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const struct sockaddr *p_serv_addr, socklen_t addrlen) Connect to socket. More...
ssp_err_t(*	socketDisconnect)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no) Disconnect socket. More...
ssp_err_t(*	socketSend)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const uint8_t *p_data, uint32_t length, uint32_t timeout_ms) Send data to connected socket. More...
ssp_err_t(*	socketRecv)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint8_t *const p_data, uint32_t length, uint32_t timeout_ms) Receive data from connected socket. More...
ssp_err_t(*	socketStatusGet)(sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint32_t *p_socket_status) Get Socket status. More...

Detailed Description

Socket Interface API

Field Documentation

◆ close

ssp_err_t(* sf_wifi_qca4010_socket_api_t::close) (sf_wifi_qca4010_socket_ctrl_t *const p_ctrl)

Close lower level driver.

Parameters

[in,out]	p_ctrl	Pointer to the control block

◆ open

ssp_err_t(* sf_wifi_qca4010_socket_api_t::open) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, sf_wifi_qca4010_socket_cfg_t const *const p_cfg)

Open lower level driver.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to configuration structure.

◆ socketConnect

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketConnect) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const struct sockaddr *p_serv_addr, socklen_t addrlen)

Connect to socket.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number
[in]	p_serv_addr	IP address to connect
[in]	addrlen	Size of socket address structure

◆ socketCreate

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketCreate) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, sf_wifi_socket_type_t type, uint8_t ipversion)

Create a socket.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number
[in]	type	TCP/UDP socket
[in]	ipversion	Protocol version

◆ socketDisconnect

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketDisconnect) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no)

Disconnect socket.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number

◆ socketRecv

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketRecv) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint8_t *const p_data, uint32_t length, uint32_t timeout_ms)

Receive data from connected socket.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number
[out]	p_data	Data Receive buffer
[in]	length	Data length to be received
[in]	timeout_ms	timeout in milliseconds

◆ socketSend

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketSend) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const uint8_t *p_data, uint32_t length, uint32_t timeout_ms)

Send data to connected socket.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number
[in]	p_data	send buffer
[in]	length	Data length to be sent
[in]	timeout_ms	timeout in milliseconds

◆ socketStatusGet

ssp_err_t(* sf_wifi_qca4010_socket_api_t::socketStatusGet) (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint32_t *p_socket_status)

Get Socket status.

Parameters

[in]	p_ctrl	pointer to control block
[in]	socket_no	Socket ID number
[out]	p_socket_status	Pointer to an integer to hold the socket return status

◆ versionGet

ssp_err_t(* sf_wifi_qca4010_socket_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	pointer to memory location to return version number

---

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_socket_api.h

sf_wifi_qca4010_socket_instance_t Struct Reference

Renesas Synergy Software Package Reference » Framework Interfaces » SF Socket WIFI Framework Interface

#include <sf_wifi_qca4010_socket_api.h>

Data Fields

sf_wifi_qca4010_socket_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_wifi_qca4010_socket_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sf_wifi_qca4010_socket_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010_socket_api.h

5.1.2.54 SF WIFI NSAL on NetX

Renesas Synergy Software Package Reference » Framework Interfaces

NetX NSAL interface implementation header file. More...

Functions

ssp_err_t	nsal_netx_driver (NX_IP_DRIVER *driver, sf_wifi_instance_t const

*p_wifi_instance, sf_wifi_nsal_cfg_t *p_wifi_nsal_cfg)

This is NetX interface driver function. More...

Detailed Description

NetX NSAL interface implementation header file.

Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI NSAL NX Framework.

Function Documentation

◆ nsal_netx_driver()

ssp_err_t nsal_netx_driver (NX_IP_DRIVER * *p_driver*, sf_wifi_instance_t const * *p_wifi_instance*, sf_wifi_nsal_cfg_t * *p_wifi_nsal_cfg*)

This is NetX interface driver function.

NSAL NetX Driver Entry function

Parameters

[in]	p_driver	NetX IP driver pointer.
[in]	p_wifi_instance	WiFi Instance.
[in]	p_wifi_nsal_cfg	NSAL configuration.

Return values

SSP_ERR_ASSERTION	WiFi Frameowrk Instance pointer or NSAL configuration pointer is NULL
SSP_SUCCESS	No NULL arguments passed and NetX driver invoked successfully

Check if the link is up

5.1.2.55 BLE Framework Interface on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces

RTOS-integrated BLE Interface Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces: More...

Modules

BLE Alert Notification Profile Interface Framework on RL78G1D

RTOS-integrated BLE Alert Notification Profile Framework example. Implementation of RL78G1D BLE Alert Notification Profile Interface Driver. It implements the following interfaces:

BLE Blood Pressure Profile Interface Framework on RL78G1D

RTOS-integrated BLE Blood Pressure Profile Framework example. Implementation of RL78G1D BLE Blood Pressure Profile Interface Driver. It implements the following interfaces:

BLE Find Me Profile Interface Framework on RL78G1D

RTOS-integrated BLE Find Me Profile Framework example. Implementation of RL78G1D BLE Find Me Profile Interface Driver. It implements the following interfaces:

BLE HID Over GATT Profile Interface Framework on RL78G1D

RTOS-integrated BLE HID Over GATT Profile Framework example. Implementation of RL78G1D BLE HID Over GATT Profile Interface Driver. It implements the following interfaces:

BLE Heart Rate Profile Interface Framework on RL78G1D

RTOS-integrated BLE Heart Rate Profile Framework example. Implementation of RL78G1D BLE Heart Rate Profile Interface Driver. It implements the following interfaces:

BLE Health Thermometer Profile Interface Framework on RL78G1D

RTOS-integrated BLE Health Thermometer Profile Framework example. Implementation of RL78G1D BLE Health Thermometer Profile Interface Driver. It implements the following interfaces:

BLE On-Board Profile Framework Interface on RL78G1D

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Interface Driver. It implements the following interfaces:

BLE Phone Alert Status Profile Interface Framework on RL78G1D

RTOS-integrated BLE Phone Alert Status Profile Framework example. Implementation of RL78G1D BLE Phone Alert Status Profile Interface Driver. It implements the following interfaces:

	BLE Proximity Profile Interface Framework on RL78G1D RTOS-integrated BLE Proximity Profile Framework example. Implementation of RL78G1D BLE Proximity Profile Interface Driver. It implements the following interfaces:
	BLE Scan Parameters Service Profile Interface Framework on RL78G1D RTOS-integrated BLE Scan Parameters Service Profile Framework example. Implementation of RL78G1D BLE Scan Parameters Service Profile Interface Driver. It implements the following interfaces:
	BLE Time Information Profile Interface Framework on RL78G1D RTOS-integrated BLE Time Information Profile Framework example. Implementation of RL78G1D BLE Time Information Profile Interface Driver. It implements the following interfaces:

Functions

ssp_err_t	ble_rl78g1d_open (sf_ble_ctrl_t *const p_ctrl, sf_ble_cfg_t const *const p_cfg, sf_ble_rl78g1d_driver_t *p_driver_data) Initializes RL78G1D Module as per user configuration, updates the control structure with instance specific information. More...
uint8_t	sf_rble_exit_status (void) Returns status whether RL78G1D driver thread should exit or continue. More...
void	copy_driver_data (sf_ble_rl78g1d_driver_t *p_driver_data, sf_ble_cfg_t const *p_cfg, sf_ble_provisioning_t const *p_prov) Copies user provisioning information or configuration in driver buffer for later use. More...
ssp_err_t	close_rl78g1d (sf_ble_ctrl_t *const p_ctrl) Un-Initialize BLE Module, stop RBLE Thread, release thread resources which are in use. More...
ssp_err_t	rl78g1d_set_provision (sf_ble_provisioning_t const *p_prov) Configures the BLE Module as per user provided Provisioning configuration. More...

ssp_err_t	rl78g1d_read_rssi (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t *p_rssi) Read RSSI information of the connection handle passed. More...
ssp_err_t	rl78g1d_start_scan (sf_ble_rl78g1d_driver_t *p_driver_data, sf_ble_scan_t *p_scan, uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info) Starts scanning of available BLE devices and returns the list of available BLE devices. More...
ssp_err_t	rl78g1d_broadcast_enable (sf_ble_rl78g1d_driver_t *p_driver_data, sf_ble_adv_info_t const *p_adv_info) Enables the advertisement of BLE Module so that other devices can find it. More...
ssp_err_t	rl78g1d_broadcast_disable (void) Disable advertisement of BLE Module so that it does not get discovered. More...
ssp_err_t	rl78g1d_add_to_white_list (uint8_t const *p_bd_addr) Adds the BLE device address to white list so that it gets scanned by the Module. More...
ssp_err_t	rl78g1d_del_from_white_list (uint8_t const *p_bd_addr) Deletes BLE device address from white list, so that it does not get discovered. More...
ssp_err_t	rl78g1d_get_info (sf_ble_ctrl_t *const p_ctrl, sf_ble_chipset_info_t *p_chipset) Gets the BLE module information and put that information in user buffer. More...
ssp_err_t	ble_rl78g1d_connect (sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle, sf_ble_rl78g1d_driver_t *p_driver_data) Connect to a remote BLE device using the information provided by user. More...
ssp_err_t	rl78g1d_disconnect (sf_ble_conn_handle_t *p_handle)

	Initiates Disconnection procedure from remote BLE device. More...
ssp_err_t	rl78g1d_set_tx_power (sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info) Sets the transmit power for the procedure specified by the connection handle. More...
ssp_err_t	rl78g1d_bonding (sf_ble_rl78g1d_driver_t *p_driver_data, uint8_t const *p_bd_addr, sf_ble_bonding_start_t *p_bonding_start) Initiates bonding procedure with Remote BLE device with configuration provided by user. More...
ssp_err_t	rl78g1d_start_encryption (sf_ble_sm_enc_info_t const *p_enc_info) Starts encryption over the BLE link using information received during Bonding procedure. More...
ssp_err_t	rl78g1d_bonding_resp (sf_ble_conn_handle_t *p_handle, sf_ble_rl78g1d_driver_t *p_driver_data, sf_ble_bonding_response_t *p_bonding_resp) Responds to bonding request from remote device with the security information provided by user. More...
ssp_err_t	rl78g1d_authorization (sf_ble_conn_handle_t *p_handle) Indicates that the specified remote device has been authorized by user. More...
void	rl78g1d_gatt_event_callback (RBLE_GATT_EVENT *p_event) GATT Event callback. More...
ssp_err_t	rl78g1d_enable_gatt (void) Enable the GATT in RBLE stack. More...
ssp_err_t	rl78g1d_gatt_service_discovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt) GATT Service discovery. More...

ssp_err_t	rl78g1d_gatt_char_discovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt) Perform characteristics discovery used by GATT client. More...
ssp_err_t	rl78g1d_gatt_char_desc_discovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle, sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t *const p_rsp_cnt) Perform characteristics descriptor discovery used by GATT client. More...
ssp_err_t	rl78g1d_gatt_char_read (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp) Perform read characteristic used by GATT client. More...
ssp_err_t	rl78g1d_gatt_char_write (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const p_char_write_req) Perform write characteristic used by GATT client. More...
ssp_err_t	rl78g1d_gatt_char_execute_write (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag) Perform execute write on all pending write operations, used by GATT client. More...
ssp_err_t	rl78g1d_gatt_char_write_local (sf_ble_ctrl_t *const p_ctrl, uint16_t char_handle, uint16_t data_length, uint8_t *const p_data) Perform local characteristic write used by GATT server. More...
ssp_err_t	rl78g1d_gatt_send_notify (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send notification to GATT client, used by GATT server. More...
ssp_err_t	rl78g1d_gatt_send_indicate (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send indication to GATT client, used by GATT server. More...

ssp_err_t	rl78g1d_gatt_write_response (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t error_code) Send response to write operation received by GATT client, used by GATT server. More...

Detailed Description

RTOS-integrated BLE Interface Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces:

SF_BLE Framework Interface API on RL78G1D.

- SF BLE Framework Interface

Function Documentation

◆ ble_rl78g1d_connect()

ssp_err_t ble_rl78g1d_connect (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_connection_t const *const *p_conn*, sf_ble_conn_handle_t * *p_handle*, sf_ble_rl78g1d_driver_t * *p_driver_data*)

Connect to a remote BLE device using the information provided by user.

Parameters

[in]	p_ctrl	Pointer to control structure which contains driver information
[in]	p_conn	Pointer to connection information
[out]	p_handle	Pointer to connection handle
[in]	p_driver_data	Pointer to driver data where data is to be copied

Return values

SSP_SUCCESS	Connect success
SSP_ERR_BLE_FAILED	Failed to connect
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_TIMEOUT	BLE event timeout

◆ ble_rl78g1d_open()

ssp_err_t ble_rl78g1d_open (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_cfg_t const *const *p_cfg*, sf_ble_rl78g1d_driver_t * *p_driver_data*)

Initializes RL78G1D Module as per user configuration, updates the control structure with instance specific information.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to BLE configuration structure
[in]	p_driver_data	Pointer to driver data where data is to be copied

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_BLE_FAILED	Failed to initialize
SSP_ERR_ALREADY_OPEN	Device is already open
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_INTERNAL	GATT Initialization failure

Copying user configuration to driver buffer for later use

◆ close_rl78g1d()

ssp_err_t close_rl78g1d (sf_ble_ctrl_t *const *p_ctrl*)

Un-Initialize BLE Module, stop RBLE Thread, release thread resources which are in use.

Closes and Stops communication with RL78G1D BLE chip

Parameters

[in]	p_ctrl	Pointer to control structure which contains driver related information

Return values

SSP_SUCCESS	Suspend the driver functionality.
SSP_ERR_BLE_FAILED	Close failure

◆ copy_driver_data()

void copy_driver_data (sf_ble_rl78g1d_driver_t * *p_driver_data*, sf_ble_cfg_t const * *p_cfg*, sf_ble_provisioning_t const * *p_prov*)

Copies user provisioning information or configuration in driver buffer for later use.

Copies the value of BLE configuration and provisioning information into the driver buffer.

Parameters

[in]	p_driver_data	Pointer to driver data where data is to be copied
[in]	p_cfg	User pointer configuration which is to be copied to driver buffer
[in]	p_prov	Pointer BLE provisioning data

Returns

None

◆ rl78g1d_add_to_white_list()

ssp_err_t rl78g1d_add_to_white_list (uint8_t const * *p_bd_addr*)

Adds the BLE device address to white list so that it gets scanned by the Module.

Bluetooth address of peer device is added to the whiteList of Module

Parameters

[in]	p_bd_addr	Pointer to bluetooth address to be added to white list

Return values

SSP_SUCCESS	Whitelist add success
SSP_ERR_BLE_FAILED	Failed to add in white list
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_authorization()

ssp_err_t rl78g1d_authorization (sf_ble_conn_handle_t * *p_handle*)

Indicates that the specified remote device has been authorized by user.

Parameters

[in]	p_handle	Pointer to BLE handle

Return values

SSP_SUCCESS	Authorization success
SSP_ERR_BLE_FAILED	Authorization Failed

Pointer to BLE connection handle

◆ rl78g1d_bonding()

ssp_err_t rl78g1d_bonding (sf_ble_rl78g1d_driver_t * *p_driver_data*, uint8_t const * *p_bd_addr*, sf_ble_bonding_start_t * *p_bonding_start*)

Initiates bonding procedure with Remote BLE device with configuration provided by user.

Starts the bonding procedure with peer device

Parameters

[in]	p_driver_data	Pointer to private driver data structure
[in]	p_bd_addr	Pointer to ble device address from which bonding is to be started
[in]	p_bonding_start	Pointer to Bonding Start information structure

Return values

SSP_SUCCESS	Bonding start success
SSP_ERR_BLE_FAILED	Bonding start failure
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_bonding_resp()

ssp_err_t rl78g1d_bonding_resp (sf_ble_conn_handle_t * *p_handle*, sf_ble_rl78g1d_driver_t * *p_driver_data*, sf_ble_bonding_response_t * *p_bonding_resp*)

Responds to bonding request from remote device with the security information provided by user.

Parameters

[in]	p_handle	Pointer to BLE handle
[in]	p_driver_data	Pointer to driver data which contains chip related information
[in]	p_bonding_resp	Bonding Response structure

Return values

SSP_SUCCESS	Bonding response success
SSP_ERR_BLE_FAILED	Failed getting bonding response
SSP_ERR_TIMEOUT	BLE event timeout

Pointer to BLE connection handle

◆ rl78g1d_broadcast_disable()

ssp_err_t rl78g1d_broadcast_disable (void)

Disable advertisement of BLE Module so that it does not get discovered.

Disable advertisement of BLE Module.

Return values

SSP_SUCCESS	Stop the broadcast of device successfully
SSP_ERR_BLE_FAILED	Failed to broadcast disable
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_broadcast_enable()

ssp_err_t rl78g1d_broadcast_enable (sf_ble_rl78g1d_driver_t * *p_driver_data*, sf_ble_adv_info_t const * *p_adv_info*)

Enables the advertisement of BLE Module so that other devices can find it.

Starts advertisement of BLE Module.

Parameters

[in]	p_driver_data	Driver data which contains chip related information
[in]	p_adv_info	Pointer to user supplied broadcast configuration

Return values

SSP_SUCCESS	Start the broadcast of device successfully
SSP_ERR_BLE_FAILED	Failed to broadcast enable
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_INVALID_MODE	Invalid arguments/parameters

Advertisement Data length

Advertisement Data

◆ rl78g1d_del_from_white_list()

ssp_err_t rl78g1d_del_from_white_list (uint8_t const * *p_bd_addr*)

Deletes BLE device address from white list, so that it does not get discovered.

Bluetooth address of peer device is deleted from the whiteList of Module

Parameters

[in]	p_bd_addr	Pointer to BLE address to be removed from white list

Return values

SSP_SUCCESS	Whitelist delete success
SSP_ERR_BLE_FAILED	Failed to delete from white list
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_disconnect()

ssp_err_t rl78g1d_disconnect (sf_ble_conn_handle_t * *p_handle*)

Initiates Disconnection procedure from remote BLE device.

BLE device is disconnected from the peer device

Parameters

[in]	p_handle	Pointer to BLE handle

Return values

SSP_SUCCESS	Disconnect success
SSP_ERR_BLE_FAILED	Disconnect failed
SSP_ERR_TIMEOUT	BLE event timeout

Pointer to BLE connection handle

◆ rl78g1d_enable_gatt()

ssp_err_t rl78g1d_enable_gatt (void)

Enable the GATT in RBLE stack.

Return values

SSP_SUCCESS	Enable the gatt successfully
SSP_ERR_INTERNAL	Failed to enable the gatt

Create semaphore for rble operation complete

◆ rl78g1d_gatt_char_desc_discovery()

ssp_err_t rl78g1d_gatt_char_desc_discovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *start_handle*, uint16_t *end_handle*, sf_ble_char_desc_discovery_rsp_t *const *p_sf_ble_chardesc_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

Perform characteristics descriptor discovery used by GATT client.

Perform Characteristic descriptor discovery of remote BLE device and pass all discovered characteristics descriptor to user

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	start_handle	Start handle from set of handle ranges to be used in discovery
[in]	end_handle	End handle from set of handle ranges to be used in discovery
[out]	p_sf_ble_chardesc_dscv_rsp	Pointer to characteristics descriptor discovery response
[in,out]	p_rsp_cnt	Input Size specifying maximum number of characteristics descriptor discovery results which can be stored in response, output specifying number of characteristics descriptor discovery results stored in response

Return values

SSP_SUCCESS	Characteristics descriptor discovery successful
SSP_ERR_BLE_FAILED	Characteristics descriptor discovery failed

Pointer to BLE connection handle

Connection handle

API call

Wait for discovery to complete

Update response count

◆ rl78g1d_gatt_char_discovery()

ssp_err_t rl78g1d_gatt_char_discovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_discovery_req_t const *const *p_sf_ble_char_dscv_req*, sf_ble_char_discovery_rsp_t *const *p_sf_ble_char_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

Perform characteristics discovery used by GATT client.

Perform Characteristic discovery of remote BLE device and pass all discovered characteristics to user

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Pointer to Connection handle
[in]	p_sf_ble_char_dscv_req	Pointer to characteristics discovery request
[out]	p_sf_ble_char_dscv_rsp	Pointer to characteristics discovery response
[in,out]	p_rsp_cnt	Input Size specifying maximum number of characteristics discovery results which can be stored in response, output specifying number of characteristics discovery results stored in response

Return values

SSP_SUCCESS	Characteristics discovery successful
SSP_ERR_BLE_FAILED	Characteristics discovery failed
SSP_ERR_NOT_OPEN	Device Not Opened

Pointer to BLE connection handle

Connection handle

API call

Wait for discovery to complete

Update response count

◆ rl78g1d_gatt_char_execute_write()

ssp_err_t rl78g1d_gatt_char_execute_write (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_execute_write_t *execute_flag*)

Perform execute write on all pending write operations, used by GATT client.

This API writes any characteristics handle which are pending in previous write

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	execute_flag	Flag specifying whether to execute or cancel pending writes

Return values

SSP_SUCCESS	Execute write successful
SSP_ERR_BLE_FAILED	Execute write failed

Pointer to BLE connection handle

Connection handle

API Call

Wait for execute write to complete

◆ rl78g1d_gatt_char_read()

ssp_err_t rl78g1d_gatt_char_read (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_read_req_t const *const *p_char_read_req*, sf_ble_char_read_rsp_t *const *p_char_read_rsp*)

Perform read characteristic used by GATT client.

GATT client reads the characteristics value of specific characteristics handle of the connected peer device

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	p_char_read_req	Pointer to characteristic read request
[out]	p_char_read_rsp	Pointer to characteristic read response

Return values

SSP_SUCCESS	Characteristic read successful
SSP_ERR_BLE_FAILED	Characteristic read failed

Characteristic handle

Characteristic handle size

Characteristic handle count

Characteristic UUID length

Characteristic UUID

Characteristic UUID length

Characteristic UUID

Characteristic UUID count

Characteristic value offset

Characteristic handle

Characteristic handle size

Characteristic handle count

Characteristic descriptor handle

Characteristic descriptor handle size

Characteristic descriptor handle count

Characteristic descriptor value offset

Characteristic descriptor handle

Characteristic descriptor handle size

Characteristic descriptor handle count

◆ rl78g1d_gatt_char_write()

ssp_err_t rl78g1d_gatt_char_write (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_write_req_t const *const *p_char_write_req*)

Perform write characteristic used by GATT client.

Write GATT characteristics data in characteristics handle of remote BLE device

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	p_char_write_req	Pointer to characteristic write request

Return values

SSP_SUCCESS	Characteristic write successful
SSP_ERR_BLE_FAILED	Characteristic write failed

◆ rl78g1d_gatt_char_write_local()

ssp_err_t rl78g1d_gatt_char_write_local (sf_ble_ctrl_t *const *p_ctrl*, uint16_t *char_handle*, uint16_t *data_length*, uint8_t *const *p_data*)

Perform local characteristic write used by GATT server.

Update local GATT database with user provided data.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	char_handle	Characteristic handle
[in]	data_length	Length of data to write
[in]	p_data	Pointer to data

Return values

SSP_SUCCESS	Local characteristic write successful
SSP_ERR_BLE_FAILED	Local characteristic write failed

Characteristic value handle

Characteristic value length

Characteristic value

API Call

Wait for local write to complete

◆ rl78g1d_gatt_event_callback()

void rl78g1d_gatt_event_callback (RBLE_GATT_EVENT * *p_event*)

GATT Event callback.

Parameters

[in]	p_event	Event details

Return values

None	

All Primary service discovery with 16 bit UUID complete

All Primary service discovery with 128 bit UUID complete

Primary service discovery by UUID complete

Included service discovery complete

All characteristics discovery with 16 bit UUID complete

All characteristics discovery with 128 bit UUID complete

Characteristics discovery by UUID (16 bit) complete

Characteristics discovery by UUID (128 bit) complete

Switch case is broken into different functions in order to reduce the cyclomatic complexity (<=10) of the function

◆ rl78g1d_gatt_send_indicate()

ssp_err_t rl78g1d_gatt_send_indicate (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *char_handle*)

Send indication to GATT client, used by GATT server.

Send the data of CCCD indication to remote GATT client.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Pointer to Connection handle
[in]	char_handle	Characteristic handle whose value will be indicated

Return values

SSP_SUCCESS	Send Indication successful
SSP_ERR_BLE_FAILED	Send Indication failed

Connection handle

Characteristic handle

API Call

Wait for indication send to complete

◆ rl78g1d_gatt_send_notify()

ssp_err_t rl78g1d_gatt_send_notify (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *char_handle*)

Send notification to GATT client, used by GATT server.

Send the data of CCCD notification to remote GATT client.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	char_handle	Characteristic handle whose value will be notified

Return values

SSP_SUCCESS	Send notification successful
SSP_ERR_BLE_FAILED	Send notification failed

Pointer to BLE connection handle

Connection handle

Characteristic handle

API Call

◆ rl78g1d_gatt_service_discovery()

ssp_err_t rl78g1d_gatt_service_discovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_service_discovery_req_t const *const *p_sf_ble_svc_dscv_req*, sf_ble_service_discovery_rsp_t *const *p_sf_ble_svc_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

GATT Service discovery.

Perform GATT service discovery of remote BLE device and pass all discovered Services to user

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	pointer to Connection handle
[in]	p_sf_ble_svc_dscv_req	Pointer to service discovery request
[out]	p_sf_ble_svc_dscv_rsp	Pointer to service discovery response
[in]	p_rsp_cnt	Input Size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

Return values

SSP_SUCCESS	Service discovery successful
SSP_ERR_BLE_FAILED	Service discovery failed
SSP_ERR_NOT_OPEN	Device Not Opened

Pointer to BLE connection handle

Connection handle

API call

Wait for discovery to complete

Update response count

◆ rl78g1d_gatt_write_response()

ssp_err_t rl78g1d_gatt_write_response (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *handle*, sf_ble_attribute_error_code_t *error_code*)

Send response to write operation received by GATT client, used by GATT server.

Parameters

[in]	p_ctrl	Pointer to control structure
[in]	p_handle	Pointer to Connection handle
[in]	handle	Characteristic handle used for write operation
[in]	error_code	Characteristic write operation error code to be sent in response

Return values

SSP_SUCCESS	Send write response successful
SSP_ERR_BLE_FAILED	Send write response failed

Pointer to BLE connection handle

Connection handle

Characteristic handle

API Call

◆ rl78g1d_get_info()

ssp_err_t rl78g1d_get_info (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_chipset_info_t * *p_chipset*)

Gets the BLE module information and put that information in user buffer.

Reads information from RL78G1D device and updates user buffer

Parameters

[in]	p_ctrl	Pointer to control structure which contains driver information
[in]	p_chipset	Pointer to user buffer which will be filled with BLE information

Return values

SSP_SUCCESS	Successfully get the BLE information
SSP_ERR_BLE_FAILED	Failed to get information from chipset
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_read_rssi()

ssp_err_t rl78g1d_read_rssi (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t * *p_rssi*)

Read RSSI information of the connection handle passed.

Reads the RSSI information of BLE module for the connection handle passed

Parameters

[in]	p_ctrl	Pointer to control structure which contains driver information
[in]	p_handle	Pointer to BLE handle
[in]	p_rssi	Pointer to user buffer in which rssi is read

Return values

SSP_SUCCESS	Successfully read rssi
SSP_ERR_BLE_FAILED	Failed to read rssi value
SSP_ERR_TIMEOUT	BLE event timeout

Pointer to BLE connection handle

◆ rl78g1d_set_provision()

ssp_err_t rl78g1d_set_provision (sf_ble_provisioning_t const * *p_prov*)

Configures the BLE Module as per user provided Provisioning configuration.

Sets bonding and security modes as provisioned.

Parameters

[in]	p_prov	Pointer to user provisioning information

Return values

SSP_SUCCESS	Successfully provisioned the device.
SSP_ERR_BLE_FAILED	Failed to set provision
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_UNSUPPORTED	Unsupported feature

Bonding mode

Variable to store temporary Bonding mode

◆ rl78g1d_set_tx_power()

ssp_err_t rl78g1d_set_tx_power (sf_ble_conn_handle_t *const *p_handle*, sf_ble_set_tx_pwr_info_t * *p_tx_power_info*)

Sets the transmit power for the procedure specified by the connection handle.

Parameters

[in]	p_handle	Pointer to BLE handle
[in]	p_tx_power_info	Pointer to TX power information

Return values

SSP_SUCCESS	TX power set successfully
SSP_ERR_BLE_FAILED	TX power set failed
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_INVALID_ARGUMENT	Invalid transmit power level parameter not supported by device

Pointer to BLE connection handle

◆ rl78g1d_start_encryption()

ssp_err_t rl78g1d_start_encryption (sf_ble_sm_enc_info_t const * *p_enc_info*)

Starts encryption over the BLE link using information received during Bonding procedure.

Parameters

[in]	p_enc_info	BLE Start Encryption information

Return values

SSP_SUCCESS	Successfully when encryption start over the BLE link
SSP_ERR_BLE_FAILED	Failed to start encryption
SSP_ERR_TIMEOUT	BLE event timeout

◆ rl78g1d_start_scan()

ssp_err_t rl78g1d_start_scan (sf_ble_rl78g1d_driver_t * *p_driver_data*, sf_ble_scan_t * *p_scan*, uint8_t * *p_cnt*, sf_ble_scan_info_t * *p_scan_info*)

Starts scanning of available BLE devices and returns the list of available BLE devices.

Starts Scanning of BLE devices either in Passive or Active mode with user provided scan configuration

Parameters

[in]	p_driver_data	Pointer to driver data where data is to be copied
[in,out]	p_scan	Pointer to user supplied scan buffer
[in,out]	p_cnt	Pointer to count variable which is to be updated with number of BLE devices scanned
[in]	p_scan_info	Pointer to scan information structure

Return values

SSP_SUCCESS	Successful scan of available BLE devices
SSP_ERR_BLE_FAILED	Failed to scan
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_INVALID_MODE	Invalid arguments/parameters

In order to start observation or to start passive scan go for observation procedure. Passive scan is only supported through observation procedure

◆ sf_rble_exit_status()

uint8_t sf_rble_exit_status (void)

Returns status whether RL78G1D driver thread should exit or continue.

Return values

SF_BLE_TRUE	RBLE Thread should exit
SF_BLE_FALSE	RBLE Thread should continue execution

BLE Alert Notification Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Alert Notification Profile Framework example. Implementation of RL78G1D BLE Alert Notification Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Alert Notification Profile Framework example. Implementation of RL78G1D BLE Alert Notification Profile Interface Driver. It implements the following interfaces:

- SF BLE Alert Notification Profile Framework Interface

BLE Blood Pressure Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Blood Pressure Profile Framework example. Implementation of RL78G1D BLE Blood Pressure Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Blood Pressure Profile Framework example. Implementation of RL78G1D BLE Blood Pressure Profile Interface Driver. It implements the following interfaces:

- SF BLE Blood Pressure Profile Framework Interface

BLE Find Me Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Find Me Profile Framework example. Implementation of RL78G1D BLE Find Me Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Find Me Profile Framework example. Implementation of RL78G1D BLE Find Me Profile Interface Driver. It implements the following interfaces:

- SF BLE Find Me Profile Framework Interface

BLE HID Over GATT Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE HID Over GATT Profile Framework example. Implementation of RL78G1D BLE HID Over GATT Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE HID Over GATT Profile Framework example. Implementation of RL78G1D BLE HID Over GATT Profile Interface Driver. It implements the following interfaces:

- SF BLE HID Over GATT Profile Framework Interface

BLE Heart Rate Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Heart Rate Profile Framework example. Implementation of RL78G1D BLE Heart Rate Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Heart Rate Profile Framework example. Implementation of RL78G1D BLE Heart Rate Profile Interface Driver. It implements the following interfaces:

- SF BLE Heart Rate Profile Framework Interface

BLE Health Thermometer Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Health Thermometer Profile Framework example. Implementation of RL78G1D BLE Health Thermometer Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Health Thermometer Profile Framework example. Implementation of RL78G1D BLE Health Thermometer Profile Interface Driver. It implements the following interfaces:

- SF BLE Health Thermometer Profile Framework Interface

BLE On-Board Profile Framework Interface on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Interface Driver. It implements the following interfaces:

- SF BLE On-Board Profile Framework Interface

BLE Phone Alert Status Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Phone Alert Status Profile Framework example. Implementation of RL78G1D BLE Phone Alert Status Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Phone Alert Status Profile Framework example. Implementation of RL78G1D BLE Phone Alert Status Profile Interface Driver. It implements the following interfaces:

- SF BLE Phone Alert Status Profile Framework Interface

BLE Proximity Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Proximity Profile Framework example. Implementation of RL78G1D BLE Proximity Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Proximity Profile Framework example. Implementation of RL78G1D BLE Proximity Profile Interface Driver. It implements the following interfaces:

- SF BLE Proximity Profile Framework Interface

BLE Scan Parameters Service Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Scan Parameters Service Profile Framework example. Implementation of RL78G1D BLE Scan Parameters Service Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Scan Parameters Service Profile Framework example. Implementation of RL78G1D BLE Scan Parameters Service Profile Interface Driver. It implements the following interfaces:

- SF BLE Scan Parameters Service Profile Framework Interface

BLE Time Information Profile Interface Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Interfaces » BLE Framework Interface on RL78G1D

RTOS-integrated BLE Time Information Profile Framework example. Implementation of RL78G1D BLE Time Information Profile Interface Driver. It implements the following interfaces: More...

RTOS-integrated BLE Time Information Profile Framework example. Implementation of RL78G1D BLE Time Information Profile Interface Driver. It implements the following interfaces:

- SF BLE Time Information Profile Framework Interface

5.1.2.56 Cellular Framework Example using Quectel CATM1 API

Renesas Synergy Software Package Reference » Framework Interfaces

SF_CELLULAR Framework API on Quectel CATM1. More...

Functions

ssp_err_t	SF_CELLULAR_QCTLCATM1_Open (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg) Initialize Cellular Quectel CATM1 Cellular driver. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_Close (sf_cellular_ctrl_t *const p_ctrl) Stop Cellular QuecTel CATM1 driver functionality. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_ProvisioningSet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning) Sets the provisioning information. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_NetworkStatusGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status)

	Get Network Status information. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_InfoGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t *const p_cellular_info) Get Cellular module information. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_VersionGet (ssp_version_t *const p_version) Get driver version based on compile time macros. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_Reset (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_reset_type_t reset_type) Reset the module. More...
ssp_err_t	sf_cellular_qctlcatm1_config_set (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *p_cfg) Set Cellular device interface configuration. More...
ssp_err_t	sf_cellular_qctlcatm1_get_imsi (sf_cellular_instance_cfg_t *p_celr_instance, uint8_t *p_imsi) Read IMSI ID. More...

Detailed Description

SF_CELLULAR Framework API on Quectel CATM1.

Function Documentation

◆ SF_CELLULAR_QCTLCATM1_Close()

ssp_err_t SF_CELLULAR_QCTLCATM1_Close (sf_cellular_ctrl_t *const *p_ctrl*)

Stop Cellular QuecTel CATM1 driver functionality.

Implements sf_cellular_api_t::close This function deactivates the PDP context and de-initializes the lower level interface

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

close module

Set module open flag and delete mutex

Delete is used in close API where all resources are released, hence no need to check the return code

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ sf_cellular_qctlcatm1_config_set()

ssp_err_t sf_cellular_qctlcatm1_config_set (sf_cellular_ctrl_t * *p_ctrl*, sf_cellular_cfg_t const * *p_cfg*)

Set Cellular device interface configuration.

Parameters

[in]	p_ctrl	Pointer to cellular control block
[in]	p_cfg	Pointer to Cellular configuration

Return values

SSP_SUCCESS	Success
SSP_ERR_CELLULAR_CONFIG_FAILED	Failed Cellular configuration
SSP_ERR_CELLULAR_INIT_FAILED	Failed due to invalid response received from modem

Local configuration variable

Set Preferred Operators List

Set Cellular operator select mode, in case of manual set operator details

Set cellular SIM priority effect. Due to the unavailability of SIM priority effect command in all the BG96 firmwares do not check the return value of this API

set cellular network fallback sequence

Set TimeZone update mode policy

◆ sf_cellular_qctlcatm1_get_imsi()

ssp_err_t sf_cellular_qctlcatm1_get_imsi (sf_cellular_instance_cfg_t * *p_celr_instance*, uint8_t * *p_imsi*)

Read IMSI ID.

Parameters

[in]	p_celr_instance	Pointer to cellular instance
[out]	p_imsi	IMSI ID

Return values

SSP_SUCCESS	Read IMSI successfully
SSP_ERR_CELLULAR_FAILED	Reading IMSI failed

AT Command used AT+CIMI - Read IMSI

Get response wait time in ticks

Send command to read IMEI number

Expected bytes of successful response

Clear response buffer

Read command response

Ignore first 2 bytes of resp_buff which contains newline ascii character

Reset result value

Delay for next try

◆ SF_CELLULAR_QCTLCATM1_InfoGet()

ssp_err_t SF_CELLULAR_QCTLCATM1_InfoGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_info_t *const *p_cellular_info*)

Get Cellular module information.

Implements sf_cellular_api_t::infoGet Get Cellular module information like chipset/driver information, RSSI, noise level, link quality

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_info	Cellular information structure

Return values

SSP_SUCCESS	Successfully get the Cellular information
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed reading Cellular information
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_QCTLCATM1_NetworkStatusGet()

ssp_err_t SF_CELLULAR_QCTLCATM1_NetworkStatusGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_network_status_t * *p_network_status*)

Get Network Status information.

Implements sf_cellular_api_t::networkStatusGet

Parameters

[in]	p_ctrl	Cellular control block
[out]	p_network_status	Cellular network structure

Return values

SSP_SUCCESS	Successfully read the Network status information
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed reading Network Status information.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_QCTLCATM1_Open()

ssp_err_t SF_CELLULAR_QCTLCATM1_Open (sf_cellular_ctrl_t * *p_ctrl*, sf_cellular_cfg_t const *const *p_cfg*)

Initialize Cellular Quectel CATM1 Cellular driver.

Implements sf_cellular_api_t::open Initializes driver and configures module with given parameters and saves this configuration.

Parameters

[out]	p_ctrl	Cellular control block
[in]	p_cfg	Cellular configuration structure

Return values

SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular QuecTel CATM1 Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular QuecTel CATM1 module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular QuecTel CATM1 module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in use

Create Mutex for Synchronization

Get Mutex Lock

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_QCTLCATM1_ProvisioningSet()

ssp_err_t SF_CELLULAR_QCTLCATM1_ProvisioningSet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_provisioning_t const *const *p_cellular_provisioning*)

Sets the provisioning information.

Implements sf_cellular_api_t::provisioningSet Sets Cellular's provisioning information

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_provisioning	Cellular provisioning structure

Return values

SSP_SUCCESS	Successfully set the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Provisioning configuration failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Set AirPlane Mode on

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_QCTLCATM1_Reset()

ssp_err_t SF_CELLULAR_QCTLCATM1_Reset (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_reset_type_t *reset_type*)

Reset the module.

Implements sf_cellular_api_t::reset This function reset the module as per the reset type

Parameters

[in]	p_ctrl	Cellular control block
[in]	reset_type	Type of reset

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command
SSP_ERR_CELLULAR_INIT_FAILED	Failed due to invalid response received from modem

Get Mutex

Reset module

Check whether SIM Pin is required

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_QCTLCATM1_VersionGet()

ssp_err_t SF_CELLULAR_QCTLCATM1_VersionGet (ssp_version_t *const *p_version*)

Get driver version based on compile time macros.

Implements sf_cellular_api_t::versionGet.

Parameters

[out]	p_version	Common version structure

Return values

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

5.1.2.57 BSD Socket over Quectel CATM1 on-chip stack API

Renesas Synergy Software Package Reference » Framework Interfaces

SF_CELLULAR Socket Framework API on Quectel CATM1. More...

Functions

ssp_err_t	SF_CELLULAR_QCTLCATM1_SOCKET_Open (sf_cellular_socket_ctrl_t *p_ctrl, sf_cellular_socket_cfg_t const *const p_cfg) Open the Cellular Device driver to use the Socket Layer on Cellular Driver On-Chip stack. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_SOCKET_Close (sf_cellular_socket_ctrl_t *const p_ctrl) Close the Cellular Device driver. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_SOCKET_PING (sf_cellular_socket_ctrl_t *const p_ctrl, ULONG *p_ip_address, uint32_t count, uint32_t interval_ms)
int	socket (int domain, int type, int protocol) This creates socket for communication. More...
int	close (int sockfd)

	This closes socket. More...
int	bind (int sockfd, const struct sockaddr *p_local_sock_addr, socklen_t addrlen) This binds socket to IP address. More...
int	listen (int sockfd, int backlog) This listens for connection on socket. More...
int	connect (int sockfd, const struct sockaddr *p_serv_addr, socklen_t addrlen) This connects with remote socket(stream socket). More...
int	accept (int sockfd, struct sockaddr *p_cliaddr, socklen_t *p_addrlen) This accepts connection from remote socket. More...
ssize_t	send (int sockfd, const void *p_buf, size_t length, int flags) This sends data over STREAM socket. More...
ssize_t	recv (int sockfd, void *p_buf, size_t length, int flags) This receives data over STREAM socket. More...
ssize_t	sendto (int sockfd, const void *p_buf, size_t length, int flags, const struct sockaddr *p_dest_addr, socklen_t addrlen) This sends data over DGRAM socket. More...
ssize_t	recvfrom (int sockfd, void *p_buf, size_t length, int flags, struct sockaddr *p_src_addr, socklen_t *p_addrlen) This receives data over DGRAM socket. More...
int	setsockopt (int sockfd, int level, int optname, const void *p_optval, socklen_t optlen) This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...

int	getsockopt (int sockfd, int level, int optname, void *p_optval, socklen_t *p_optlen) This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission. More...
int	select (int nfds, fd_set *p_readfds, fd_set *p_writefds, fd_set *p_exceptfds, struct timeval *p_timeout) This waits for any activity on socket. More...
ssp_err_t	SF_CELLULAR_QCTLCATM1_SOCKET_VersionGet (ssp_version_t *const p_version) This function gets the version information of the underlying driver. More...

Detailed Description

SF_CELLULAR Socket Framework API on Quectel CATM1.

Function Documentation

◆ accept()

int accept (int *sockfd*, struct sockaddr * *p_cliaddr*, socklen_t * *p_addrlen*)

This accepts connection from remote socket.

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network address
[out]	p_addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error accepting the connection
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Connection is received successfully.

◆ bind()

int bind (int *sockfd*, const struct sockaddr * *p_local_sock_addr*, socklen_t *addrlen*)

This binds socket to IP address.

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Binding socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is bound successfully.

◆ close()

int close (int *sockfd*)

This closes socket.

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Closing socket failed
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is closed successfully.

◆ connect()

int connect (int *sockfd*, const struct sockaddr * *p_serv_addr*, socklen_t *addrlen*)

This connects with remote socket(stream socket).

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error occurred.
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket is connected successfully.

◆ getsockopt()

int getsockopt (int *sockfd*, int *level*, int *optname*, void * *p_optval*, socklen_t * *p_optlen*)

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error reading socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option read successfully.

◆ listen()

int listen (int *sockfd*, int *backlog*)

This listens for connection on socket.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Set socket in Listen mode successfully.

◆ recv()

ssize_t recv (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*)

This receives data over STREAM socket.

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to receive data.

Returns

Otherwise Number of Data bytes received successfully.

◆ recvfrom()

ssize_t recvfrom (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*, struct sockaddr * *p_src_addr*, socklen_t * *p_addrlen*)

This receives data over DGRAM socket.

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address
[in]	p_addrlen	Size of Remote machine network

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error receiving data

Returns

Otherwise Numbers of data bytes received successfully.

◆ select()

int select (int *nfds*, fd_set * *p_readfds*, fd_set * *p_writefds*, fd_set * *p_exceptfds*, struct timeval * *p_timeout*)

This waits for any activity on socket.

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set. If no descriptor is to be tested for writing, p_writefds should be NULL
[in]	p_exceptfds	Exceptional socket fd set. If no descriptor is to be tested for exceptions, p_exceptfds should be NULL
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Timeout occurred, no activity.

Returns

Otherwise Activity detected(Packet available).

◆ send()

ssize_t send (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*)

This sends data over STREAM socket.

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Failed to send data.

Returns

Otherwise Number of Data bytes sent successfully.

◆ sendto()

ssize_t sendto (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*, const struct sockaddr * *p_dest_addr*, socklen_t *addrlen*)

This sends data over DGRAM socket.

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data
[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error Sending data.

Returns

Otherwise Numbers of bytes sent successfully.

◆ setsockopt()

int setsockopt (int *sockfd*, int *level*, int *optname*, const void * *p_optval*, socklen_t *optlen*)

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Invalid input arguments OR Error setting socket option
SF_CELLULAR_QCTLCATM1_SOCKET_SUCCESS	Socket option set successfully.

◆ SF_CELLULAR_QCTLCATM1_SOCKET_Close()

ssp_err_t SF_CELLULAR_QCTLCATM1_SOCKET_Close (sf_cellular_socket_ctrl_t *const *p_ctrl*)

Close the Cellular Device driver.

Implements sf_cellular_socket_api_t::close Calls the low level Cellular device driver's Close API to close the cellular Driver.

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in use

SF Cellular instance variable

SF Cellular API structure variable

SF Cellular Stack control block structure

◆ SF_CELLULAR_QCTLCATM1_SOCKET_Open()

ssp_err_t SF_CELLULAR_QCTLCATM1_SOCKET_Open (sf_cellular_socket_ctrl_t * *p_ctrl*, sf_cellular_socket_cfg_t const *const *p_cfg*)

Open the Cellular Device driver to use the Socket Layer on Cellular Driver On-Chip stack.

Implements sf_cellular_socket_api_t::open Calls the low level Cellular device driver's Open API to Initialize the Cellular Device Driver, for using Socket Layer on On-Chip stack.

Return values

SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular Quectel CATM1 Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular Quectel CATM1 module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular Quectel CATM1 module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in use

SF Cellular instance variable

SF Cellular API structure variable

SF Cellular configuration structure variable

SF Cellular Stack control block structure

Initialize Cellular Modem

Initialize Socket Interface global variable , if module open successfully or already open

◆ SF_CELLULAR_QCTLCATM1_SOCKET_PING()

ssp_err_t SF_CELLULAR_QCTLCATM1_SOCKET_PING (sf_cellular_socket_ctrl_t *const *p_ctrl*, ULONG * *p_ip_address*, uint32_t *count*, uint32_t *interval_ms*)

Ping an IP address on the network.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_ip_address	Pointer to IP address.
[in]	count	Number of ping attempts.
[in]	interval_ms	Interval between ping attempts.

Return values

SSP_ERR_UNSUPPORTED	Functionality not supported

◆ SF_CELLULAR_QCTLCATM1_SOCKET_VersionGet()

ssp_err_t SF_CELLULAR_QCTLCATM1_SOCKET_VersionGet (ssp_version_t *const *p_version*)

This function gets the version information of the underlying driver.

Implements sf_cellular_socket_api_t::versionGet

Parameters

[out]	p_version	Driver version information

Return values

SSP_ERR_ASSERTION	Argument NULL is passed
SSP_SUCCESS	Successfully read version information

◆ socket()

int socket (int *domain*, int *type*, int *protocol*)

This creates socket for communication.

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_QCTLCATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket descriptor of newly created socket

5.1.2.58 Cellular Framework Example using RYZ014CATM1 API

Renesas Synergy Software Package Reference » Framework Interfaces

SF_CELLULAR Framework API on RYZ014CATM1. More...

Functions

ssp_err_t	SF_CELLULAR_RYZ014CATM1_Open (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg) Initialize Cellular RYZ014 CATM1 Cellular driver. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_Close (sf_cellular_ctrl_t *const p_ctrl) Stop Cellular RYZ014 CATM1 driver functionality. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_ProvisioningSet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning) Sets the provisioning information. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_InfoGet (sf_cellular_ctrl_t *const p_ctrl,

	sf_cellular_info_t *const p_cellular_info) Get Cellular module information. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_VersionGet (ssp_version_t *const p_version) Get driver version based on compile time macros. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_Reset (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_reset_type_t reset_type) Reset the module. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_NetworkStatusGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status) Get Network Status information. More...
ssp_err_t	sf_cellular_ryz014catm1_config_set (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *p_cfg) Set Cellular device interface configuration. More...
ssp_err_t	sf_cellular_ryz014catm1_set_pdp_context_state (sf_cellular_instance_cfg_t *p_celr_instance, uint8_t cid, uint8_t activate) Activate/Deactivate the PDP context. More...
ssp_err_t	sf_cellular_ryz014catm1_get_imsi (sf_cellular_instance_cfg_t *p_celr_instance, uint8_t *p_imsi) Read IMSI ID. More...

Detailed Description

SF_CELLULAR Framework API on RYZ014CATM1.

Function Documentation

◆ SF_CELLULAR_RYZ014CATM1_Close()

ssp_err_t SF_CELLULAR_RYZ014CATM1_Close (sf_cellular_ctrl_t *const *p_ctrl*)

Stop Cellular RYZ014 CATM1 driver functionality.

Implements sf_cellular_api_t::close This function deactivates the PDP context and de-initializes the lower level interface

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

close module

Set module open flag and delete mutex

Delete is used in close API where all resources are released, hence no need to check the return code

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ sf_cellular_ryz014catm1_config_set()

ssp_err_t sf_cellular_ryz014catm1_config_set (sf_cellular_ctrl_t * *p_ctrl*, sf_cellular_cfg_t const * *p_cfg*)

Set Cellular device interface configuration.

Parameters

[in]	p_ctrl	Pointer to cellular control block
[in]	p_cfg	Pointer to Cellular configuration

Return values

SSP_SUCCESS	Success
SSP_ERR_CELLULAR_CONFIG_FAILED	Failed Cellular configuration
SSP_ERR_CELLULAR_INIT_FAILED	Failed due to invalid response received from modem

Set Preferred Operators List

Set Cellular operator select mode, in case of manual set operator details

Set TimeZone update mode policy

◆ sf_cellular_ryz014catm1_get_imsi()

ssp_err_t sf_cellular_ryz014catm1_get_imsi (sf_cellular_instance_cfg_t * *p_celr_instance*, uint8_t * *p_imsi*)

Read IMSI ID.

Parameters

[in]	p_celr_instance	Pointer to cellular instance
[out]	p_imsi	IMSI ID

Return values

SSP_SUCCESS	Read IMSI successfully
SSP_ERR_CELLULAR_FAILED	Reading IMSI failed

AT Command used AT+CIMI - Read IMSI

Get response wait time in ticks

Send command to read IMEI number

Expected bytes of successful response

Clear response buffer

Read command response

Ignore first 2 bytes of resp_buff which contains newline ascii character

Reset result value

Delay for next try

◆ SF_CELLULAR_RYZ014CATM1_InfoGet()

ssp_err_t SF_CELLULAR_RYZ014CATM1_InfoGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_info_t *const *p_cellular_info*)

Get Cellular module information.

Implements sf_cellular_api_t::infoGet Get Cellular module information like chipset/driver information, RSSI, noise level, link quality

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_info	Cellular information structure

Return values

SSP_SUCCESS	Successfully get the Cellular information
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed reading Cellular information
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_RYZ014CATM1_NetworkStatusGet()

ssp_err_t SF_CELLULAR_RYZ014CATM1_NetworkStatusGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_network_status_t * *p_network_status*)

Get Network Status information.

Implements sf_cellular_api_t::networkStatusGet

Parameters

[in]	p_ctrl	Cellular control block
[out]	p_network_status	Cellular network structure

Return values

SSP_SUCCESS	Successfully read the Network status information
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed reading Network Status information.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_RYZ014CATM1_Open()

ssp_err_t SF_CELLULAR_RYZ014CATM1_Open (sf_cellular_ctrl_t * *p_ctrl*, sf_cellular_cfg_t const *const *p_cfg*)

Initialize Cellular RYZ014 CATM1 Cellular driver.

Implements sf_cellular_api_t::open Initializes driver and configures module with given parameters and saves this configuration.

Parameters

[out]	p_ctrl	Cellular control block
[in]	p_cfg	Cellular configuration structure

Return values

SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular RYZ014 CATM1 Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular RYZ014 CATM1 module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular RYZ014 CATM1 module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in use

Create Mutex for Synchronization

Get Mutex Lock

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_RYZ014CATM1_ProvisioningSet()

ssp_err_t SF_CELLULAR_RYZ014CATM1_ProvisioningSet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_provisioning_t const *const *p_cellular_provisioning*)

Sets the provisioning information.

Implements sf_cellular_api_t::provisioningSet Sets Cellular's provisioning information

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_provisioning	Cellular provisioning structure

Return values

SSP_SUCCESS	Successfully set the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Provisioning configuration failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Set AirPlane Mode on

Set provisioning

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_RYZ014CATM1_Reset()

ssp_err_t SF_CELLULAR_RYZ014CATM1_Reset (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_reset_type_t *reset_type*)

Reset the module.

Implements sf_cellular_api_t::reset This function reset the module as per the reset type

Parameters

[in]	p_ctrl	Cellular control block
[in]	reset_type	Type of reset

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command
SSP_ERR_CELLULAR_INIT_FAILED	Failed due to invalid response received from modem

Get Mutex

Reset module

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ sf_cellular_ryz014catm1_set_pdp_context_state()

ssp_err_t sf_cellular_ryz014catm1_set_pdp_context_state (sf_cellular_instance_cfg_t * *p_celr_instance*, uint8_t *cid*, uint8_t *activate*)

Activate/Deactivate the PDP context.

Parameters

[in]	p_celr_instance	Pointer to Cellular Instance configuration
[in]	cid	Content ID
[in]	activate	Context status to be set

Return values

SSP_SUCCESS	Successfully activate/deactivate the PDP context
SSP_ERR_CELLULAR_FAILED	Failed to activate/deactivate the PDP context

AT Command used AT+CGACT=socket,1 - To activate the context AT+CGACT=socket,0 - To deactivate the context AT+CGACT=Socket_context,activate or deactivate

Create socket context activation command

Expected bytes of successful response

Create context activation command

Expected bytes of successful response

Store data in string format

numbers of bytes to send

Send command

Get response wait time in ticks

Read response

Check whether expected response present or not

< In case of activation

< In case of de-activation

Delay for next retry

◆ SF_CELLULAR_RYZ014CATM1_VersionGet()

ssp_err_t SF_CELLULAR_RYZ014CATM1_VersionGet (ssp_version_t *const *p_version*)

Get driver version based on compile time macros.

Implements sf_cellular_api_t::versionGet.

Parameters

[out]	p_version	Common version structure

Return values

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

5.1.2.59 SF CELLULAR Common Interface

Renesas Synergy Software Package Reference » Framework Interfaces

SF_Cellular Framework API Common Code. More...

Functions

ssp_err_t	SF_CELLULAR_COMMON_Open (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg) Initialize Cellular driver. More...
ssp_err_t	SF_CELLULAR_COMMON_Close (sf_cellular_ctrl_t *const p_ctrl) Stop Cellular driver functionality. More...
ssp_err_t	SF_CELLULAR_COMMON_InfoGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t *const p_cellular_info) Get Cellular module information. More...
ssp_err_t	SF_CELLULAR_COMMON_StatisticsGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_stats_t *const p_cellular_device_stats) Get the interface statistics. More...
ssp_err_t	SF_CELLULAR_COMMON_Transmit (sf_cellular_ctrl_t *const p_ctrl,

	uint8_t *const p_buf, uint32_t length) Passes packet buffer to PPP stack for transmission. More...
ssp_err_t	SF_CELLULAR_COMMON_ProvisioningGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t *const p_cellular_provisioning) Reads the provisioning information. More...
ssp_err_t	SF_CELLULAR_COMMON_ProvisioningSet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning) Sets the provisioning information. More...
ssp_err_t	SF_CELLULAR_COMMON_NetworkConnect (sf_cellular_ctrl_t *const p_ctrl) Establishes the Data connection. More...
ssp_err_t	SF_CELLULAR_COMMON_NetworkConnectWithCGDATA (sf_cellular_ctrl_t *const p_ctrl) Establishes the Data connection using AT+CGDATA command. Limitation: This API activates only one PDP Context Id even though AT+CGDATA command actually supports activation of multiple Context Ids in single command. More...
ssp_err_t	SF_CELLULAR_COMMON_NetworkDisconnect (sf_cellular_ctrl_t *const p_ctrl) Terminates the connection. More...
ssp_err_t	SF_CELLULAR_COMMON_NetworkStatusGet (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status) Get Network Status information. More...
ssp_err_t	SF_CELLULAR_COMMON_SimPinSet (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_old_pin, uint8_t *const p_new_pin) Set the SIM PIN. More...
ssp_err_t	SF_CELLULAR_COMMON_SimLock (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin) Lock the SIM. More...

ssp_err_t	SF_CELLULAR_COMMON_SimUnlock (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin) Unlock the SIM. More...
ssp_err_t	SF_CELLULAR_COMMON_SimIDGet (sf_cellular_ctrl_t *const p_ctrl, uint8_t *p_sim_id) Get SIM ID. More...
ssp_err_t	SF_CELLULAR_COMMON_CommandSend (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_cmd_resp_t *const p_input_at_command, sf_cellular_cmd_resp_t *const p_output, uint32_t const timeout) Send AT command directly to Cellular Modem. More...
ssp_err_t	SF_CELLULAR_COMMON_FotaCheck (sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck) Checks for available firmware upgrade. More...
ssp_err_t	SF_CELLULAR_COMMON_FotaStart (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart) Perform the firmware upgrade. More...
ssp_err_t	SF_CELLULAR_COMMON_FotaStop (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop) Stop firmware upgrade. More...

Detailed Description

SF_Cellular Framework API Common Code.

Cellular Framework header files Framework header files for this package HAL Layer communication interface header file

Function Documentation

◆ SF_CELLULAR_COMMON_Close()

ssp_err_t SF_CELLULAR_COMMON_Close (sf_cellular_ctrl_t *const *p_ctrl*)

Stop Cellular driver functionality.

Implements sf_cellular_api_t::close This function deactivates the PDP context and Update global variables for future use

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Delete is used in close API where all resources are released, hence no need to check the return code

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_CommandSend()

ssp_err_t SF_CELLULAR_COMMON_CommandSend (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_cmd_resp_t *const *p_input_at_command*, sf_cellular_cmd_resp_t *const *p_output*, uint32_t const *timeout*)

Send AT command directly to Cellular Modem.

This API will send AT command provided by user to the Cellular Modem and will collect the response from the Modem and will send it back to the user. If Modem is in Data Mode when this API is called then Framework will first switch Modem to Command Mode, then send the AT command and collect the response and then switches the Modem back to Data Mode.

Parameters

[in]	p_ctrl	Pointer to the control block for the Cellular module.
[in]	p_input_at_command	Pointer to structure which contains Modem command to send
[in,out]	p_output	Pointer to buffer in which response will be sent to user, Also user will pass the size of the buffer which is pointed by p_output
[in]	timeout	Timeout for which framework will wait for response in milliseconds

Return values

SSP_SUCCESS	Successfully sent AT command and collected response from Cellular modem
SSP_ERR_CELLULAR_FAILED	Failed to either send AT command or collect response from Cellular modem
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_FotaCheck()

ssp_err_t SF_CELLULAR_COMMON_FotaCheck (sf_cellular_ctrl_t *const *p_ctrl*, void * *p_fotacheck*)

Checks for available firmware upgrade.

Implements sf_cellular_api_t::fotaCheck Checks for available firmware upgrade.

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_fotacheck	Fota check specific structure

Return values

SSP_ERR_UNSUPPORTED	Functionality not supported

◆ SF_CELLULAR_COMMON_FotaStart()

ssp_err_t SF_CELLULAR_COMMON_FotaStart (sf_cellular_ctrl_t *const *p_ctrl*, void * *p_fotastart*)

Perform the firmware upgrade.

Implements sf_cellular_api_t::fotaStart Perform the firmware upgrade.

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_fotastart	Fota start specific structure

Return values

SSP_ERR_UNSUPPORTED	Functionality not supported

◆ SF_CELLULAR_COMMON_FotaStop()

ssp_err_t SF_CELLULAR_COMMON_FotaStop (sf_cellular_ctrl_t *const *p_ctrl*, void * *p_fotastop*)

Stop firmware upgrade.

Implements sf_cellular_api_t::fotaStop Stop firmware upgrade

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_fotastop	Fota stop specific structure

Return values

SSP_ERR_UNSUPPORTED	Functionality not supported

◆ SF_CELLULAR_COMMON_InfoGet()

ssp_err_t SF_CELLULAR_COMMON_InfoGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_info_t *const *p_cellular_info*)

Get Cellular module information.

Implements sf_cellular_api_t::infoGet Get Cellular module information like chipset/driver information, RSSI, noise level, link quality

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_info	Cellular information structure

Return values

SSP_SUCCESS	Successfully get the Cellular information
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed reading Cellular information
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_NetworkConnect()

ssp_err_t SF_CELLULAR_COMMON_NetworkConnect (sf_cellular_ctrl_t *const *p_ctrl*)

Establishes the Data connection.

Implements sf_cellular_api_t::networkConnect Establishes the Network Connection

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Successfully establishes the Network connection
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed to establish the Network Connection
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Enter Data Mode

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_NetworkConnectWithCGDATA()

ssp_err_t SF_CELLULAR_COMMON_NetworkConnectWithCGDATA (sf_cellular_ctrl_t *const *p_ctrl*)

Establishes the Data connection using AT+CGDATA command. Limitation: This API activates only one PDP Context Id even though AT+CGDATA command actually supports activation of multiple Context Ids in single command.

Implements sf_cellular_api_t::networkConnect Establishes the Network Connection

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Successfully establishes the Network connection
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed to establish the Network Connection
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Enter Data Mode

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_NetworkDisconnect()

ssp_err_t SF_CELLULAR_COMMON_NetworkDisconnect (sf_cellular_ctrl_t *const *p_ctrl*)

Terminates the connection.

Implements sf_cellular_api_t::networkDisconnect

Parameters

[in]	p_ctrl	Cellular control block

Return values

SSP_SUCCESS	Successfully disconnect the connection
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to terminate the network connection
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Exit Data Mode

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_NetworkStatusGet()

ssp_err_t SF_CELLULAR_COMMON_NetworkStatusGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_network_status_t * *p_network_status*)

Get Network Status information.

Implements sf_cellular_api_t::networkStatusGet

Parameters

[in]	p_ctrl	Cellular control block
[out]	p_network_status	Cellular network structure

Return values

SSP_SUCCESS	Successfully read the Network status information
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed reading Network Status information.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_Open()

ssp_err_t SF_CELLULAR_COMMON_Open (sf_cellular_ctrl_t * *p_ctrl*, sf_cellular_cfg_t const *const *p_cfg*)

Initialize Cellular driver.

Implements sf_cellular_api_t::open Initializes Cellular Driver and Configure the parameters as per the p_cfg Update global variables for future use.

Parameters

[out]	p_ctrl	Cellular control block
[in]	p_cfg	Cellular configuration structure

Return values

SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in used

Create Mutex for Synchronization

Get Mutex Lock

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_ProvisioningGet()

ssp_err_t SF_CELLULAR_COMMON_ProvisioningGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_provisioning_t *const *p_cellular_provisioning*)

Reads the provisioning information.

Implements sf_cellular_api_t::provisioningGet Reads Cellular's provisioning information

Parameters

[in]	p_ctrl	Cellular control block
[out]	p_cellular_provisioning	Cellular provisioning structure

Return values

SSP_SUCCESS	Successfully read the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Reading provisioning information failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

Copy provisioning info to driver structure

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_ProvisioningSet()

ssp_err_t SF_CELLULAR_COMMON_ProvisioningSet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_provisioning_t const *const *p_cellular_provisioning*)

Sets the provisioning information.

Implements sf_cellular_api_t::provisioningSet Sets Cellular's provisioning information

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_provisioning	Cellular provisioning structure

Return values

SSP_SUCCESS	Successfully set the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Provisioning configuration failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command
SSP_ERR_CELLULAR_INIT_FAILED	Failed due to invalid response received from modem

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_SimIDGet()

ssp_err_t SF_CELLULAR_COMMON_SimIDGet (sf_cellular_ctrl_t *const *p_ctrl*, uint8_t * *p_sim_id*)

Get SIM ID.

Implements sf_cellular_api_t::simGetID Get SIM ID

Parameters

[in]	p_ctrl	Cellular control block
[out]	p_sim_id	SIM ID

Return values

SSP_SUCCESS	Successfully get the SIM ID.
SSP_ERR_CELLULAR_FAILED	Failed to get the SIM ID
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_SimLock()

ssp_err_t SF_CELLULAR_COMMON_SimLock (sf_cellular_ctrl_t *const *p_ctrl*, uint8_t *const *p_pin*)

Lock the SIM.

Implements sf_cellular_api_t::simLock

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_pin	SIM Pin

Return values

SSP_SUCCESS	Successfully lock the SIM
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to Lock the SIM
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_SimPinSet()

ssp_err_t SF_CELLULAR_COMMON_SimPinSet (sf_cellular_ctrl_t *const *p_ctrl*, uint8_t *const *p_old_pin*, uint8_t *const *p_new_pin*)

Set the SIM PIN.

Implements sf_cellular_api_t::simSetPin

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_old_pin	Old SIM Pin
[in]	p_new_pin	New SIM Pin

Return values

SSP_SUCCESS	Successfully set SIM Pin
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to set the SIM Pin
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_SimUnlock()

ssp_err_t SF_CELLULAR_COMMON_SimUnlock (sf_cellular_ctrl_t *const *p_ctrl*, uint8_t *const *p_pin*)

Unlock the SIM.

Implements sf_cellular_api_t::simUnlock

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_pin	SIM Pin

Return values

SSP_SUCCESS	Successfully unlock the SIM
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to unlock the SIM
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

◆ SF_CELLULAR_COMMON_StatisticsGet()

ssp_err_t SF_CELLULAR_COMMON_StatisticsGet (sf_cellular_ctrl_t *const *p_ctrl*, sf_cellular_stats_t *const *p_cellular_device_stats*)

Get the interface statistics.

Implements sf_cellular_api_t::statisticsGet Collect the statistics information of Cellular interface

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_cellular_device_stats	Cellular statistic structure

Return values

SSP_SUCCESS	Successfully get the Statistics information.
SSP_ERR_UNSUPPORTED	Functionality is not supported.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed Reading statistics information
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

◆ SF_CELLULAR_COMMON_Transmit()

ssp_err_t SF_CELLULAR_COMMON_Transmit (sf_cellular_ctrl_t *const *p_ctrl*, uint8_t *const *p_buf*, uint32_t *length*)

Passes packet buffer to PPP stack for transmission.

Implements sf_cellular_api_t::transmit Send packet buffer to PPP stack

Parameters

[in]	p_ctrl	Cellular control block
[in]	p_buf	transmit byte buffer pointer
[in]	length	Length of data

Return values

SSP_SUCCESS	Successfully send the packet buffer.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Transmitting Data failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

Get Mutex

The return code is not checked here because tx_mutex_put cannot fail when called with a mutex owned by the current thread. The mutex is owned by the current thread because this call follows a successful call to tx_mutex_get.

5.1.2.60 BSD Socket over RYZ014CATM1 on-chip stack API

Renesas Synergy Software Package Reference » Framework Interfaces

SF_CELLULAR Socket Framework API on RYZ014CATM1. More...

Functions

ssp_err_t	SF_CELLULAR_RYZ014CATM1_SOCKET_Open (sf_cellular_socket_ctrl_t *p_ctrl, sf_cellular_socket_cfg_t const *const p_cfg) Open the Cellular Device driver to use the Socket Layer on Cellular Driver On-Chip stack. More...

ssp_err_t	SF_CELLULAR_RYZ014CATM1_SOCKET_Close (sf_cellular_socket_ctrl_t *const p_ctrl) Close the Cellular Device driver. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_SOCKET_PING (sf_cellular_socket_ctrl_t *const p_ctrl, ULONG *p_ip_address, uint32_t count, uint32_t interval_ms)
int	socket (int domain, int type, int protocol) This creates socket for communication. More...
int	close (int sockfd) This closes socket. More...
int	bind (int sockfd, const struct sockaddr *p_local_sock_addr, socklen_t addrlen) This binds socket to IP address. More...
int	listen (int sockfd, int backlog) This listens for connection on socket. More...
int	connect (int sockfd, const struct sockaddr *p_serv_addr, socklen_t addrlen) This connects with remote socket(stream socket). More...
int	accept (int sockfd, struct sockaddr *p_cliaddr, socklen_t *p_addrlen) This accepts connection from remote socket. More...
ssize_t	send (int sockfd, const void *p_buf, size_t length, int flags) This sends data over STREAM socket. More...
ssize_t	recv (int sockfd, void *p_buf, size_t length, int flags) This receives data over STREAM socket. More...
ssize_t	sendto (int sockfd, const void *p_buf, size_t length, int flags, const struct sockaddr *p_dest_addr, socklen_t addrlen)

	This sends data over DGRAM socket. More...
ssize_t	recvfrom (int sockfd, void *p_buf, size_t length, int flags, struct sockaddr *p_src_addr, socklen_t *p_addrlen) This receives data over DGRAM socket. More...
int	setsockopt (int sockfd, int level, int optname, const void *p_optval, socklen_t optlen) This sets socket specific options. More...
int	getsockopt (int sockfd, int level, int optname, void *p_optval, socklen_t *p_optlen) This gets socket specific options. More...
int	select (int nfds, fd_set *p_readfds, fd_set *p_writefds, fd_set *p_exceptfds, struct timeval *p_timeout) This waits for any activity on socket. More...
ssp_err_t	SF_CELLULAR_RYZ014CATM1_SOCKET_VersionGet (ssp_version_t *const p_version) This function gets the version information of the underlying driver. More...

Detailed Description

SF_CELLULAR Socket Framework API on RYZ014CATM1.

Function Documentation

◆ accept()

int accept (int *sockfd*, struct sockaddr * *p_cliaddr*, socklen_t * *p_addrlen*)

This accepts connection from remote socket.

Implements accept This function accepts connection from remote socket. This API is used only with socket type STREAM. The call is blocked if no connection is present or the socket is blocking.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_cliaddr	Remote machine Network address (Can be NULL)
[out]	p_addrlen	Size of Socket address structure(Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error accepting the connection or invalid input parameters

Returns

Otherwise Connection is received successfully.

◆ bind()

int bind (int *sockfd*, const struct sockaddr * *p_local_sock_addr*, socklen_t *addrlen*)

This binds socket to IP address.

Implements bind This function binds socket to given IP address and port.

Parameters

sockfd	Socket file descriptor
p_local_sock_addr	Socket address structure
addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Binding socket failed or invalid input parameters

Returns

Otherwise Socket is bound successfully.

◆ close()

int close (int *sockfd*)

This closes socket.

Implements close Close opened socket

Parameters

[in]	sockfd	Socket file descriptor

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Closing socket failed or invalid input parameters

Returns

Otherwise Socket is closed successfully.

◆ connect()

int connect (int *sockfd*, const struct sockaddr * *p_serv_addr*, socklen_t *addrlen*)

This connects with remote socket(stream socket).

Implements connect This function connects local socket with remote socket. The call is blocked until a connection is established or error is returned.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_serv_addr	Remote machine Network address
[in]	addrlen	Size of Socket address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error occurred or invalid input parameters

Returns

Otherwise Socket is connected successfully.

◆ getsockopt()

int getsockopt (int *sockfd*, int *level*, int *optname*, void * *p_optval*, socklen_t * *p_optlen*)

This gets socket specific options.

This reads socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements getsockopt This gets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[out]	p_optval	Socket option value
[out]	p_optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error reading socket option or invalid socket descriptor

Returns

Otherwise Socket option read successfully.

◆ listen()

int listen (int *sockfd*, int *backlog*)

This listens for connection on socket.

Implements listen This function listen for connection on socket.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	backlog	number of maximum connection can be accepted

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to set socket in Listen mode or invalid input parameters

Returns

Otherwise Set socket in Listen mode successfully.

◆ recv()

ssize_t recv (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*)

This receives data over STREAM socket.

Implements receive This function receives data on a stream socket in connected state. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket File Descriptor
[out]	p_buf	Buffer to read data
[out]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to receive data or invalid input parameters

Returns

Otherwise Data received successfully.

◆ recvfrom()

ssize_t recvfrom (int *sockfd*, void * *p_buf*, size_t *length*, int *flags*, struct sockaddr * *p_src_addr*, socklen_t * *p_addrlen*)

This receives data over DGRAM socket.

Implements recvfrom This function receives data on a dgram socket. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API will do a partial copy, and hold on the rest of the payload for a subsequent call to the API.

Parameters

[in]	sockfd	Socket file descriptor
[out]	p_buf	Data buffer pointer to read data
[in]	length	Length of data to read
[in]	flags	Flags
[in]	p_src_addr	Remote machine network address (Can be NULL)
[in]	p_addrlen	Size of Remote machine network (Can be NULL)

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error receiving data or invalid input parameters

Returns

Otherwise Numbers of data bytes received successfully.

◆ select()

int select (int *nfds*, fd_set * *p_readfds*, fd_set * *p_writefds*, fd_set * *p_exceptfds*, struct timeval * *p_timeout*)

This waits for any activity on socket.

Implements select Allow an application thread to block on a given socket handle for a specified time period. This API checks for any activity on specified socket, for example arrival of a packet at the receive queue.

Parameters

[in]	nfds	Number of socket fds in to check
[in]	p_readfds	Read socket fd set
[in]	p_writefds	Write socket fd set - API does not used this parameter
[in]	p_exceptfds	Exceptional socket fd set - API does not used this parameter
[in]	p_timeout	Timeout

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Timeout occurred, no activity or invalid socket descriptor

Returns

Otherwise Activity detected(Packet available).

◆ send()

ssize_t send (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*)

This sends data over STREAM socket.

Implements send This function sends data on a stream socket in connected state.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data to send
[in]	length	Data length
[in]	flags	Socket flag

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Failed to send data or invalid input parameters

Returns

Otherwise Data sent successfully.

◆ sendto()

ssize_t sendto (int *sockfd*, const void * *p_buf*, size_t *length*, int *flags*, const struct sockaddr * *p_dest_addr*, socklen_t *addrlen*)

This sends data over DGRAM socket.

Implements sendto This function sends data to remote.

Parameters

[in]	sockfd	Socket File Descriptor
[in]	p_buf	Buffer data pointer
[in]	length	Length of data
[in]	flags	Flag
[in]	p_dest_addr	Address of remote UDP server
[in]	addrlen	Size of Network address structure

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error Sending data or invalid input parameters

Returns

Otherwise Numbers of bytes sent successfully.

◆ setsockopt()

int setsockopt (int *sockfd*, int *level*, int *optname*, const void * *p_optval*, socklen_t *optlen*)

This sets socket specific options.

This updates socket specific options. Quectel CATM1 supports following socket options SO_SNDBUF: Transmission packet size TCP_MAX_RETRIES: Maximum Number of retransmission TCP_MAX_RTO: Maximum interval time of TCP retransmission.

Implements setsockopt This sets options for an existing socket.

Parameters

[in]	sockfd	Socket file descriptor
[in]	level	Level
[in]	optname	Socket option name
[in]	p_optval	Socket option value
[in]	optlen	Size of Socket option

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Error setting socket option or invalid socket descriptor

Returns

Otherwise Socket option set successfully.

◆ SF_CELLULAR_RYZ014CATM1_SOCKET_Close()

ssp_err_t SF_CELLULAR_RYZ014CATM1_SOCKET_Close (sf_cellular_socket_ctrl_t *const *p_ctrl*)

Close the Cellular Device driver.

Implements sf_cellular_socket_api_t::close Calls the low level Cellular device driver's Close API to close the cellular Driver.

Return values

SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in used

SF Cellular instance variable

SF Cellular API structure variable

SF Cellular Stack control block structure

Close cellular framework

◆ SF_CELLULAR_RYZ014CATM1_SOCKET_Open()

ssp_err_t SF_CELLULAR_RYZ014CATM1_SOCKET_Open (sf_cellular_socket_ctrl_t * *p_ctrl*, sf_cellular_socket_cfg_t const *const *p_cfg*)

Open the Cellular Device driver to use the Socket Layer on Cellular Driver On-Chip stack.

Implements sf_cellular_socket_api_t::open Calls the low level Cellular device driver's Open API to Initialize the Cellular Device Driver, for using Socket Layer on On-Chip stack.

Return values

SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular RYZ014CATM1 Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular RYZ014CATM1 module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular RYZ014CATM1 module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in used

SF Cellular instance variable

SF Cellular API structure variable

SF Cellular configuration structure variable

SF Cellular Stack control block structure

Initialize Cellular Modem

Initialize Socket Interface global variable

◆ SF_CELLULAR_RYZ014CATM1_SOCKET_PING()

ssp_err_t SF_CELLULAR_RYZ014CATM1_SOCKET_PING (sf_cellular_socket_ctrl_t *const *p_ctrl*, ULONG * *p_ip_address*, uint32_t *count*, uint32_t *interval_ms*)

Ping an IP address on the network.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_ip_address	Pointer to IP address.
[in]	count	Number of ping attempts.
[in]	interval_ms	Interval between ping attempts.

Return values

SSP_SUCCESS	Function completed successfully.
SSP_ERR_CELLULAR_FAILED	Driver initialization failed.
SSP_ERR_IN_USE	Device already in use

Get mutex lock

Ping specified IP address

Release the mutex

◆ SF_CELLULAR_RYZ014CATM1_SOCKET_VersionGet()

ssp_err_t SF_CELLULAR_RYZ014CATM1_SOCKET_VersionGet (ssp_version_t *const *p_version*)

This function gets the version information of the underlying driver.

Implements sf_cellular_socket_api_t::versionGet

Parameters

[out]	p_version	Driver version information

Return values

SSP_ERR_ASSERTION	Argument NULL is passed
SSP_SUCCESS	Successfully read version information

◆ socket()

int socket (int *domain*, int *type*, int *protocol*)

This creates socket for communication.

Implements socket Creates socket with given parameters.

Parameters

[in]	domain	Network Domain
[in]	type	Socket type
[in]	protocol	Protocol type

Return values

SF_CELLULAR_RYZ014CATM1_SOCKET_INVALID_FD	Socket creation failed

Returns

Otherwise Socket created successfully

5.1.3 Framework Layer

Renesas Synergy Software Package Reference

Modules

ADC periodic Framework

RTOS-integrated ADC Framework.

Audio Framework

RTOS-integrated Audio Framework.

DAC Audio Playback Framework

RTOS-integrated DAC implementation of Audio Playback Interface.

I2S Audio Playback Framework

RTOS-integrated I2S implementation of Audio Playback Interface.

ADC Audio recording Framework

RTOS-integrated ADC implementation of Audio Recording Interface.

	I2S Audio recording Framework I2S implementation of Audio Recording Interface.
	BLOCK_MEDIA_LEVELX_NOR RTOS-integrated Block Media framework for LEVELX driver.
	BLOCK_MEDIA_QSPI RTOS-integrated Block Media framework for QSPI driver.
	BLOCK_MEDIA_RAM RTOS-integrated Block Media framework for RAM.
	BLOCK_MEDIA_SDMMC RTOS-integrated Block Media framework for SDMMC driver.
	Cellular NSAL Implementation on NetX Cellular NetX NSAL interface implementation header file.
	Telnet Communication Framework on sf_comms_telnet RTOS-integrated Communications Framework NetX telnet server implementation.
	Console Framework RTOS-integrated Console Framework.
	SSP Crypto Common Framework RTOS-integrated Crypto Common Framework Module.
	SSP Crypto Cipher Framework RTOS-integrated Crypto Cipher Framework Module.
	SSP Crypto Hash Framework RTOS-integrated Crypto HASH Framework Module.

	SSP Crypto Key Framework RTOS-integrated Crypto Key Framework Module.
	SSP Crypto Key Installation Framework RTOS-integrated Crypto Key Installation Framework Module.
	SSP Crypto Signature Framework RTOS-integrated Crypto Signature Framework Module.
	SSP Crypto TRNG Framework RTOS-integrated Crypto TRNG Framework Module.
	FX_IO Framework FileX adaptation layer for block media device drivers.
	GUIX Synergy Port GUIX adaptation layer.
	EL_LX_NOR LevelX NOR driver implementation.
	USB Communication Framework V2 RTOS-integrated USBX CDC ACM device implementation.
	External IRQ Framework RTOS-integrated external IRQ Framework.
	I2C Framework RTOS-integrated I2C Framework.
	JPEG Framework RTOS-integrated JPEG Framework.

	Memory framework RTOS-integrated Memory framework for QSPI NOR driver.
	Messaging Framework RTOS-integrated Messaging Framework implementation.
	Power Profiles Framework V2 Power Profiles Framework.
	SPI Framework RTOS-integrated SPI Framework.
	Thread Monitor Framework Framework module providing monitoring of threads.
	CTSU V2 Framework CTSU V2 Framework.
	Touch Panel V2 Framework RTOS-integrated touch panel V2 Framework implementation touch chips.
	UART Framework Instance RTOS-integrated Communications Framework UART implementation.
	NetX Synergy Port RTOS-integrated NetX Ethernet driver for the Renesas Synergy software and Synergy Ethernet IP.
	NetX Synergy Port PHY Driver Interface between SF_EL_NX Ethernet framework and PHY driver.
	BLE Framework on RL78G1D RTOS-integrated BLE Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces:

BLE On-Board Profile Framework on RL78G1D

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Driver. It implements the following interfaces:

Cellular Framework Example using Quectel CATM1

RTOS-integrated Cellular Framework example. Implementation of Cellular Quectel CATM1 Driver. It implements the following interfaces:

BSD Socket over Quectel CATM1 on-chip stack

RTOS-integrated Cellular Socket Framework example. Implementation of Quectel CATM1 Socket layer over Quectel CATM1 On-Chip stack It implements the following interfaces:

Cellular Framework Example using RYZ014 CATM1

RTOS-integrated Cellular Framework example. Implementation of Cellular RYZ014 CATM1 Driver. It implements the following interfaces:

BSD Socket over RYZ014CATM1 on-chip stack

RTOS-integrated Cellular Socket Framework example. Implementation of RYZ014CATM1 Socket layer over RYZ014CATM1 On-Chip stack It implements the following interfaces:

Touch Panel Framework Example for FT5X06

RTOS-integrated touch panel chip ft5x06 example. Implementation of ft5x06 touch chip Driver. It implements the following interfaces:

Touch Panel Framework Example for SX8654

RTOS-integrated touch panel chip sx8654 example. Implementation of sx8654 touch chip Driver. It implements the following interfaces:

WiFi Framework on GT202

RTOS-integrated WiFi Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces:

WiFi On Chip Stack on GT202

	RTOS-integrated WiFi On Chip Stack Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces:
	BSD Socket on GT202 Implementation of GT202 Socket layer over GT202 On-Chip stack.
	WiFi Framework on QCA4010 RTOS-integrated WiFi Framework example. Implementation of Silex ULPGN WiFi Driver. It implements the following interfaces:
	WiFi On Chip Stack on QCA4010 RTOS-integrated WiFi On Chip Stack Framework example. Implementation of SILEX WiFi Driver. It implements the following interfaces:
	Socket on QCA4010 Implementation of QCA4010 Socket layer over QCA4010 On-Chip stack.
	USBX Framework RTOS-integrated USBX adaptation framework for Synergy. Implements USB HOST and DEVICE low level device drivers.
	2D Drawing Engine Support Framework

Detailed Description

The framework layer provides RTOS aware drivers for functional use cases.

5.1.3.1 ADC periodic Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated ADC Framework. More...

Data Structures

struct	sf_adc_periodic_instance_ctrl_t

Macros

#define	SF_ADC_PERIODIC_CODE_VERSION_MAJOR (2U)

Functions

void	sf_adc_periodic_demo_task (ULONG thread_input)
void	app_callback (sf_adc_periodic_callback_args_t *p_args)
ssp_err_t	SF_ADC_PERIODIC_Open (sf_adc_periodic_ctrl_t *const p_api_ctrl, sf_adc_periodic_cfg_t const *const p_cfg) Configures periodic ADC framework and optionally starts the timer. More...
ssp_err_t	SF_ADC_PERIODIC_Start (sf_adc_periodic_ctrl_t *const p_api_ctrl) Gets mutex, starts the periodic ADC scan, and releases mutex. More...
ssp_err_t	SF_ADC_PERIODIC_Stop (sf_adc_periodic_ctrl_t *const p_api_ctrl) Gets mutex, stops the periodic ADC scan, and releases mutex. More...
ssp_err_t	SF_ADC_PERIODIC_Close (sf_adc_periodic_ctrl_t *const p_api_ctrl) The close function acquires the unit's mutex, closes all lower level drivers, releases and deletes the mutex. More...
ssp_err_t	SF_ADC_PERIODIC_VersionGet (ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

RTOS-integrated ADC Framework.

Macro Definition Documentation

◆ SF_ADC_PERIODIC_CODE_VERSION_MAJOR

#define SF_ADC_PERIODIC_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ app_callback()

void app_callback (sf_adc_periodic_callback_args_t * *p_args*)

Callback function that will be called when the requested number of sampling iterations are complete

If the event indicates that new data is available

Assuming only one channel is configured, data for that channel is available via the buffer index. Refer to the usage manual for more details.

◆ SF_ADC_PERIODIC_Close()

ssp_err_t SF_ADC_PERIODIC_Close (sf_adc_periodic_ctrl_t *const *p_api_ctrl*)

The close function acquires the unit's mutex, closes all lower level drivers, releases and deletes the mutex.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.

Get mutex since this will access hardware registers

Close the HAL layer modules

Delete RTOS services used

Clear information from control block so other functions know this instance is closed

◆ sf_adc_periodic_demo_task()

void sf_adc_periodic_demo_task (ULONG *thread_input*)
Initialize the framework
Check for error condition
Start the scan process
Check for error condition
Stop the scan process
Check for error condition
Restart the scan process
Check for error condition
Stop the scan process
Check for error condition
Close the framework
Check for error condition

◆ SF_ADC_PERIODIC_Open()

ssp_err_t SF_ADC_PERIODIC_Open (sf_adc_periodic_ctrl_t *const *p_api_ctrl*, sf_adc_periodic_cfg_t const *const *p_cfg*)

Configures periodic ADC framework and optionally starts the timer.

The SF_ADC_PERIODIC_Open function acquires a mutex for the ADC Unit used, then calls the driver .open function in the p_api parameter. The mutex is released following the driver layer open function.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl or p_cfg. See HAL driver for other possible causes.
SSP_ERR_INVALID_ARGUMENT	An invalid argument is used
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Save driver structure pointer for use in other framework layer functions

Create a mutex

Initialize the HAL layer

If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code

Delete the mutex.

Mark control block open so other tasks know it is valid

◆ SF_ADC_PERIODIC_Start()

ssp_err_t SF_ADC_PERIODIC_Start (sf_adc_periodic_ctrl_t *const *p_api_ctrl*)

Gets mutex, starts the periodic ADC scan, and releases mutex.

Warning

The driver will enable the ADC to be triggered via timer event; there will be a time delay from the time this function is called to the time the hardware timer count expires and triggers the scan.

Return values

SSP_SUCCESS	ADC Periodic Scan started successfully.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_IN_USE	The module is currently busy performing another operation

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- adc_api_t::scanStart
- timer_api_t::start

Get mutex, start timer, then release mutex

Enable the ADC to receive hardware triggers

If the scan was successfully enabled in the ADC HAL,

Start the timer to generate the ADC trigger events

Return the mutex.

◆ SF_ADC_PERIODIC_Stop()

ssp_err_t SF_ADC_PERIODIC_Stop (sf_adc_periodic_ctrl_t *const *p_api_ctrl*)

Gets mutex, stops the periodic ADC scan, and releases mutex.

Return values

SSP_SUCCESS	Periodic ADC scan stopped successfully.
SSP_ERR_ASSERTION	One or more pointers point to NULL..
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_IN_USE	The module is currently busy performing another operation

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- timer_api_t::stop

Get mutex, stop timer, then release mutex

Return the mutex.

◆ SF_ADC_PERIODIC_VersionGet()

ssp_err_t SF_ADC_PERIODIC_VersionGet (ssp_version_t *const *p_version*)

Gets version and stores it in provided pointer p_version.

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

sf_adc_periodic_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » ADC periodic Framework

#include <sf_adc_periodic.h>

Data Fields

uint32_t	open Used by driver to check if pointer to control block is valid.
TX_MUTEX	mutex Mutex used to protect access to lower level driver hardware registers.
adc_instance_t const *	p_lower_lvl_adc Pointer to the ADC instance.
timer_instance_t const *	p_lower_lvl_timer Pointer to the Timer instance.
transfer_instance_t const *	p_lower_lvl_transfer Pointer to the Transfer instance.
void const *volatile	p_src_transfer Source pointer for the low level transfer method.
uint16_t *	p_data_buffer Pointer to the buffer that will store the samples.
uint32_t	data_buffer_length Length of the data buffer that will store the samples.
uint32_t	data_buffer_index Index of the data buffer where data is to be written to next.
uint32_t	sample_count Samples per channel to be buffered before notifying the app.
uint32_t	current_sample_count Current sample count of the output buffer.

uint32_t	dtc_transfer_length Total Length of DTC transfer for requested number of samples.
uint32_t	size_multiplier Multiplier used to treat p_data_buffer as a 32-bit array.
uint16_t	length Length of transfer.
bool	lower_level Used to detect lower level driver.
void(*	p_callback)(sf_adc_periodic_callback_args_t *p_args) Callback function.
void const *	p_context Placeholder for user data.

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when SF_ADC_PERIODIC_Open is called

---

The documentation for this struct was generated from the following file:

- sf_adc_periodic.h

5.1.3.2 Audio Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Audio Framework. More...

Data Structures

struct	sf_audio_playback_common_instance_ctrl_t

struct	sf_audio_playback_instance_ctrl_t

Macros

#define	SF_AUDIO_PLAYBACK_CODE_VERSION_MAJOR (2U)
#define	SF_AUDIO_PLAYBACK_STACK_SIZE (SF_AUDIO_PLAYBACK_CFG_THREAD_STACK_SIZE)

Functions

ssp_err_t	SF_AUDIO_PLAYBACK_Open (sf_audio_playback_ctrl_t *const p_api_ctrl, sf_audio_playback_cfg_t const *const p_cfg)
ssp_err_t	SF_AUDIO_PLAYBACK_Close (sf_audio_playback_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_Start (sf_audio_playback_ctrl_t *const p_api_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)
ssp_err_t	SF_AUDIO_PLAYBACK_Pause (sf_audio_playback_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_Stop (sf_audio_playback_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_Resume (sf_audio_playback_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_VolumeSet (sf_audio_playback_ctrl_t *const p_api_ctrl, uint8_t const volume)
ssp_err_t	SF_AUDIO_PLAYBACK_VersionGet (ssp_version_t *const p_version)

Detailed Description

RTOS-integrated Audio Framework.

Summary

This module is a ThreadX-aware Audio Framework. The module implements Audio Framework Interface.

Name of module used by error logger macro

Macro Definition Documentation

◆ SF_AUDIO_PLAYBACK_CODE_VERSION_MAJOR

#define SF_AUDIO_PLAYBACK_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ SF_AUDIO_PLAYBACK_STACK_SIZE

#define SF_AUDIO_PLAYBACK_STACK_SIZE (SF_AUDIO_PLAYBACK_CFG_THREAD_STACK_SIZE)
Audio playback internal thread stack size. Varies by application, but rarely requires more than 256 bytes.

Function Documentation

◆ SF_AUDIO_PLAYBACK_Close()

ssp_err_t SF_AUDIO_PLAYBACK_Close (sf_audio_playback_ctrl_t *const *p_api_ctrl*)

Implements sf_audio_playback_api_t::close.

Return values

SSP_SUCCESS	Audio instance successfully closed
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is not reentrant.

Mark stream as unused in hardware control block and determine if all streams are closed.

Mark hardware control block as unused so it can be reconfigured.

Close lower level drivers

Delete RTOS services used

Mark control block as unused so it can be reconfigured.

◆ SF_AUDIO_PLAYBACK_Open()

ssp_err_t SF_AUDIO_PLAYBACK_Open (sf_audio_playback_ctrl_t *const *p_api_ctrl*, sf_audio_playback_cfg_t const *const *p_cfg*)

Implements sf_audio_playback_api_t::open.

Return values

SSP_SUCCESS	Audio hardware successfully configured.
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is not reentrant.

Open hardware if it is not already open.

Enter a critical section before checking the common instance mutex status.

Check if common instance mutex is already created. If not then create the mutex.

Create common_instance_mutex to protect common initialization, including initialization of shared event flags, audio playback thread and lower level hardware.

If mutex create fails, return error.

Exit critical section

Acquire the mutex before accessing the shared resource. Try again if the mutex was deleted in close.

Create event flags to notify playback thread when playback of a buffer is complete.

Store stream pointer in common control block.

Release the mutex.

Mark stream opened so it can be used by other API's.

◆ SF_AUDIO_PLAYBACK_Pause()

ssp_err_t SF_AUDIO_PLAYBACK_Pause (sf_audio_playback_ctrl_t *const *p_api_ctrl*)

Implements sf_audio_playback_api_t::pause.

Return values

SSP_SUCCESS	Audio playback pause message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant.

Send message with pause event to audio thread.

◆ SF_AUDIO_PLAYBACK_Resume()

ssp_err_t SF_AUDIO_PLAYBACK_Resume (sf_audio_playback_ctrl_t *const *p_api_ctrl*)

Implements sf_audio_playback_api_t::resume.

Return values

SSP_SUCCESS	Audio playback resume message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Note

This function is reentrant.

Send message with resume event to audio thread.

◆ SF_AUDIO_PLAYBACK_Start()

ssp_err_t SF_AUDIO_PLAYBACK_Start (sf_audio_playback_ctrl_t *const *p_api_ctrl*, sf_audio_playback_data_t *const *p_data*, UINT const *timeout*)

Implements sf_audio_playback_api_t::start.

Return values

SSP_SUCCESS	Buffer successfully sent to audio playback thread
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Note

This function is reentrant.

Ensure that audio data is only posted from the current stream owner.

Store new stream owner if stream is unowned.

Set message header to audio start event. Set instance to stream instance.

Send message with audio data to audio thread.

◆ SF_AUDIO_PLAYBACK_Stop()

ssp_err_t SF_AUDIO_PLAYBACK_Stop (sf_audio_playback_ctrl_t *const *p_api_ctrl*)

Implements sf_audio_playback_api_t::stop.

Return values

SSP_SUCCESS	Audio playback stop message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant.

Send message with stop event to audio thread.

◆ SF_AUDIO_PLAYBACK_VersionGet()

ssp_err_t SF_AUDIO_PLAYBACK_VersionGet (ssp_version_t *const *p_version*)

Implements sf_audio_playback_api_t::versionGet.

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

◆ SF_AUDIO_PLAYBACK_VolumeSet()

ssp_err_t SF_AUDIO_PLAYBACK_VolumeSet (sf_audio_playback_ctrl_t *const *p_api_ctrl*, uint8_t const *volume*)

Implements sf_audio_playback_api_t::volumeSet.

Return values

SSP_SUCCESS	Audio playback software volume level updated (applies to all streams).
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

Update volume in control block.

sf_audio_playback_common_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Audio Framework

#include <sf_audio_playback.h>

Public Member Functions

uint8_t stack [SF_AUDIO_PLAYBACK_STACK_SIZE]	BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)

Data Fields

uint32_t	open Used to determine if driver is initialized.

void const *	p_next_buffer Pointer to next buffer (to be played when the current buffer completes).
uint32_t	next_length Length of next buffer (to be played when the current buffer completes).
sf_message_instance_t const *	p_message Pointer to message control block.
TX_QUEUE *	p_queue Queue subscribed to SF_MESSAGE_EVENT_CLASS_AUDIO events.
sf_audio_playback_hw_instance_t const *	p_lower_lvl_hw Hardware API's used.
sf_audio_playback_instance_ctrl_t *	p_stream [SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS] Stream specific data.
TX_THREAD	thread Main audio thread.
TX_EVENT_FLAGS_GROUP	flags Event flags used to end wait in audio thread.
sf_audio_playback_data_type_t	data_type Sample format required by the hardware.
uint8_t	volume Volume from 0 (muted) to 255 (maximum, default on open).

uint8_t	buffer_index Which ping pong buffer to use.
int16_t	samples [2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]
volatile bool	playing State of audio instance (currently playing if true)
TX_MUTEX	common_instance_mutex Mutex for internal use.

Detailed Description

Audio common instance control block. DO NOT INITIALIZE. Initialization the first time sf_audio_playback_api_t::open is called. Shared by all streams.

Member Function Documentation

◆ BSP_ALIGN_VARIABLE_V2()

uint8_t stack [SF_AUDIO_PLAYBACK_STACK_SIZE]
sf_audio_playback_common_instance_ctrl_t::BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)

Stack for audio thread.

Field Documentation

◆ samples

int16_t sf_audio_playback_common_instance_ctrl_t::samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]

Ping pong buffers, used to store converted data during transfer.

The documentation for this struct was generated from the following file:

- sf_audio_playback.h

sf_audio_playback_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Audio Framework

#include <sf_audio_playback.h>

Data Fields

uint32_t	open Used to determine if driver is initialized.
TX_THREAD *	p_owner
void(*	p_callback)(sf_message_callback_args_t *p_args)
uint8_t	class_instance Class instance used to identify the stream to the messaging framework.
uint32_t	samples_remaining Internal state of data samples remaining for this stream.
uint32_t	samples_total Total number of samples to play (independent of sample size like 8/12/16).
uint32_t	index Internal state of current data index for this stream.
uint32_t	end Used to track completion of looped playback.
sf_audio_playback_data_t *	p_data [2] Audio data read from queue.
sf_audio_playback_status_t	status Status of current stream.

sf_audio_playback_common_instance_ctrl_t *	p_common_ctrl

Detailed Description

Audio stream instance control block. DO NOT INITIALIZE. Initialization occurs when sf_audio_playback_api_t::open is called.

Field Documentation

◆ p_callback

void(* sf_audio_playback_instance_ctrl_t::p_callback) (sf_message_callback_args_t *p_args)
Callback called when playback of a buffer passed to sf_audio_playback_api_t::start is complete.

◆ p_common_ctrl

sf_audio_playback_common_instance_ctrl_t* sf_audio_playback_instance_ctrl_t::p_common_ctrl
Pointer to the hardware control block used by this stream.

◆ p_owner

TX_THREAD* sf_audio_playback_instance_ctrl_t::p_owner
Pointer to thread that began the stream at this index. Used to ensure multiple threads don't interleave data on the same stream.

The documentation for this struct was generated from the following file:

- sf_audio_playback.h

5.1.3.3 DAC Audio Playback Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated DAC implementation of Audio Playback Interface. More...

Data Structures

struct	sf_audio_playback_hw_dac_instance_ctrl_t

struct	sf_audio_playback_hw_dac_cfg_t

Functions

ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_Open (sf_audio_playback_hw_ctrl_t *const p_api_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_Start (sf_audio_playback_hw_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_Stop (sf_audio_playback_hw_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_Play (sf_audio_playback_hw_ctrl_t *const p_api_ctrl, int16_t const *const p_buffer, uint32_t length)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_Close (sf_audio_playback_hw_ctrl_t *const p_api_ctrl)
ssp_err_t	SF_AUDIO_PLAYBACK_HW_DAC_VersionGet (ssp_version_t *const p_version)

Variables

sf_audio_playback_hw_api_t	g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

Detailed Description

RTOS-integrated DAC implementation of Audio Playback Interface.

The Audio Playback Framework DAC implementation uses a timer to generate events at the sampling frequency, and uses these events to transfer PCM samples to the DAC.

Function Documentation

◆ SF_AUDIO_PLAYBACK_HW_DAC_Close()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Close (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*)

Close open audio driver.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- timer_api_t::close
- dac_api_t::close
- transfer_api_t::close

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Close timer driver.

Close DAC driver.

Close transfer driver.

◆ SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet (sf_audio_playback_hw_ctrl_t *const *p_ctrl*, sf_audio_playback_data_type_t *const *p_data_type*)

Provides the expected data type in the pointer p_data_type.

Return values

SSP_SUCCESS	Data type stored in p_data_type.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_data_type is NULL.

Note

This function is reentrant if the lower level driver functions are reentrant.

Store data type. The Synergy DAC supports only 12-bit unsigned data.

◆ SF_AUDIO_PLAYBACK_HW_DAC_Open()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Open (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*, sf_audio_playback_hw_cfg_t const *const *p_cfg*)

Open the DAC audio driver, including the DAC HAL driver and helper timer and transfer HAL drivers.

Return values

SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	Null Pointer.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- dac_api_t::open
- timer_api_t::open
- transfer_api_t::open
- timer_api_t::close
- dac_api_t::close

Note

This function is reentrant if the lower level driver functions are reentrant.

Open Timer driver at selected frequency

If DTC is selected, register the audio callback with the timer ISR. DTC calls activation source ISR when the transfer is complete.

Open DAC module

Open transfer module to transfer from buffer to DAC output register

Configure transfer size of driver depending upon the underlying DAC resolution

Open transfer driver

Store driver data.

Play linear ramp data to get DAC up to half the maximum output.

◆ SF_AUDIO_PLAYBACK_HW_DAC_Play()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Play (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*, int16_t const *const *p_buffer*, uint32_t *length*)

Play a single audio buffer by input samples to the DAC at the sampling frequency configured by the timer.

Return values

SSP_SUCCESS	Buffer playback began successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_buffer is NULL or length is greater than 0x10000UL.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- transfer_api_t::reset

Note

This function is reentrant if the lower level driver functions are reentrant.

Reset transfer.

◆ SF_AUDIO_PLAYBACK_HW_DAC_Start()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Start (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*)

Start the DAC and timer HAL drivers.

Return values

SSP_SUCCESS	Audio playback hardware started successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- dac_api_t::start
- timer_api_t::start

Note

This function is reentrant if the lower level driver functions are reentrant.

Start DAC.

Start timer.

◆ SF_AUDIO_PLAYBACK_HW_DAC_Stop()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Stop (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*)

Stop the DAC and timer HAL drivers.

Return values

SSP_SUCCESS	Audio playback hardware stopped successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls
- timer_api_t::stop
- dac_api_t::stop

Note

This function is reentrant if the lower level driver functions are reentrant.

Stop timer.

Stop DAC.

◆ SF_AUDIO_PLAYBACK_HW_DAC_VersionGet()

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_VersionGet (ssp_version_t *const *p_version*)

Stores the version of the firmware and API in provided pointer p_version.

Return values

SSP_ERR_ASSERTION	The parameter p_version is NULL.
SSP_SUCCESS	Module version successfully stored in p_version.

Note

This function is reentrant.

Variable Documentation

◆ g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

sf_audio_playback_hw_api_t g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

Function pointers for DAC implementation of audio playback API.

sf_audio_playback_hw_dac_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » DAC Audio Playback Framework

#include <sf_audio_playback_hw_dac.h>

Data Fields

void(*	p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
void *	p_context
dac_instance_t const *	p_lower_lvl_dac DAC API used to access DAC hardware.
timer_instance_t const *	p_lower_lvl_timer Timer API used to generate sampling frequency.
transfer_instance_t const *	p_lower_lvl_transfer Transfer API used to transfer data each sampling frequency.
volatile bool	is_dac_ramped_up Whether the DAC is ramped up to half of maximum output.

Detailed Description

Hardware dependent control block for DAC audio driver.

Field Documentation

◆ p_callback

void(* sf_audio_playback_hw_dac_instance_ctrl_t::p_callback) (sf_audio_playback_hw_callback_args_t *p_args)
Callback called when play is complete.

◆ p_context

void* sf_audio_playback_hw_dac_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_dac.h

sf_audio_playback_hw_dac_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » DAC Audio Playback Framework

#include <sf_audio_playback_hw_dac.h>

Data Fields

dac_instance_t const *	p_lower_lvl_dac DAC API used to access DAC hardware.
timer_instance_t const *	p_lower_lvl_timer Timer API used to generate sampling frequency.
transfer_instance_t const *	p_lower_lvl_transfer Transfer API used to transfer data each sampling frequency.

Detailed Description

Hardware dependent configuration for DAC audio driver.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_dac.h

5.1.3.4 I2S Audio Playback Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated I2S implementation of Audio Playback Interface. More...

Data Structures

struct	sf_audio_playback_hw_i2s_instance_ctrl_t
struct	sf_audio_playback_hw_i2s_cfg_t

Functions

ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_Open (sf_audio_playback_hw_ctrl_t *const p_api_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg) Open the I2S audio driver, including the I2S HAL driver and helper timer and transfer HAL drivers. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_Start (sf_audio_playback_hw_ctrl_t *const p_ctrl) Start the I2S and timer HAL drivers. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_Stop (sf_audio_playback_hw_ctrl_t *const p_api_ctrl) Stop the I2S and timer HAL drivers. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_Play (sf_audio_playback_hw_ctrl_t *const p_api_ctrl, int16_t const *const p_buffer, uint32_t length) Play a single audio buffer by input samples to the I2S at the sampling frequency configured by the timer. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type) Provides the expected data type in the pointer p_data_type. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_Close (sf_audio_playback_hw_ctrl_t *const p_api_ctrl) Close open audio driver. More...
ssp_err_t	SF_AUDIO_PLAYBACK_HW_I2S_VersionGet (ssp_version_t *const p_version) Stores the version of the firmware and API in provided pointer p_version. More...

Detailed Description

RTOS-integrated I2S implementation of Audio Playback Interface.

The Audio Playback Framework I2S implementation uses the I2S interface for audio playback.

Name of module used by error logger macro

Function Documentation

◆ SF_AUDIO_PLAYBACK_HW_I2S_Close()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Close (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*)

Close open audio driver.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Returns

See Common Error Codes and lower level driver function for other possible return codes.
This function calls:
- i2s_api_t::close

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Close I2S driver.

◆ SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet (sf_audio_playback_hw_ctrl_t *const *p_ctrl*, sf_audio_playback_data_type_t *const *p_data_type*)

Provides the expected data type in the pointer p_data_type.

Return values

SSP_SUCCESS	Data type stored in p_data_type.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_data_type is NULL.

Note

This function is reentrant if the lower level driver functions are reentrant.

Store data type. The audio framework supports only 16-bit signed data.

◆ SF_AUDIO_PLAYBACK_HW_I2S_Open()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Open (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*, sf_audio_playback_hw_cfg_t const *const *p_cfg*)

Open the I2S audio driver, including the I2S HAL driver and helper timer and transfer HAL drivers.

Return values

SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	One of the following parameter is null: p_ctrl or p_cfg or p_cfg_extend->p_lower_lvl_i2s or p_cfg_extend->p_lower_lvl_i2s->p_api.

Returns

See Common Error Codes and lower level driver function for other possible return codes.
This function calls:
- i2s_api_t::open

Note

This function is reentrant if the lower level driver functions are reentrant.

Open I2S module

Store driver data.

◆ SF_AUDIO_PLAYBACK_HW_I2S_Play()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Play (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*, int16_t const *const *p_buffer*, uint32_t *length*)

Play a single audio buffer by input samples to the I2S at the sampling frequency configured by the timer.

Return values

SSP_SUCCESS	Buffer playback began successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_buffer is NULL or length is less than 0x10000U.

Returns

See Common Error Codes and lower level driver function for other possible return codes.
This function calls:
- i2s_api_t::write

Note

This function is reentrant if the lower level driver functions are reentrant.

Reset transfer.

◆ SF_AUDIO_PLAYBACK_HW_I2S_Start()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Start (sf_audio_playback_hw_ctrl_t *const *p_ctrl*)

Start the I2S and timer HAL drivers.

Return values

SSP_SUCCESS	Audio playback hardware started successfully.

Note

This function is reentrant if the lower level driver functions are reentrant.

This API is not used - I2S is started when write is called from SF_AUDIO_PLAYBACK_HW_I2S_Play.

◆ SF_AUDIO_PLAYBACK_HW_I2S_Stop()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Stop (sf_audio_playback_hw_ctrl_t *const *p_api_ctrl*)

Stop the I2S and timer HAL drivers.

Return values

SSP_SUCCESS	Audio playback hardware stopped successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Returns

See Common Error Codes and lower level driver function for other possible return codes.
This function calls:
- i2s_api_t::stop

Note

This function is reentrant if the lower level driver functions are reentrant.

Stop I2S.

◆ SF_AUDIO_PLAYBACK_HW_I2S_VersionGet()

ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_VersionGet (ssp_version_t *const *p_version*)

Stores the version of the firmware and API in provided pointer p_version.

Return values

SSP_ERR_ASSERTION	The parameter p_version is NULL.
SSP_SUCCESS	Module version successfully stored in p_version.

Note

This function is reentrant.

sf_audio_playback_hw_i2s_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » I2S Audio Playback Framework

#include <sf_audio_playback_hw_i2s.h>

Data Fields

void(*	p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
void *	p_context
i2s_instance_t const *	p_lower_lvl_i2s I2S API used to access I2S hardware.

Detailed Description

Hardware dependent control block for I2S audio driver.

Field Documentation

◆ p_callback

void(* sf_audio_playback_hw_i2s_instance_ctrl_t::p_callback) (sf_audio_playback_hw_callback_args_t *p_args)
Callback called when play is complete.

◆ p_context

void* sf_audio_playback_hw_i2s_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_i2s.h

sf_audio_playback_hw_i2s_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » I2S Audio Playback Framework

#include <sf_audio_playback_hw_i2s.h>

Data Fields

i2s_instance_t const *	p_lower_lvl_i2s I2S API used to access I2S hardware.

Detailed Description

Hardware dependent configuration for I2S audio driver.

The documentation for this struct was generated from the following file:

- sf_audio_playback_hw_i2s.h

5.1.3.5 ADC Audio recording Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated ADC implementation of Audio Recording Interface. More...

Data Structures

struct	sf_audio_record_adc_instance_ctrl_t Control block for audio recording Initialization occurs when sf_audio_record_api_t::open is called. More...

Macros

#define	SF_AUDIO_RECORD_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_AUDIO_RECORD_ADC_Open (sf_audio_record_ctrl_t *const p_api_ctrl, sf_audio_record_cfg_t const *const p_cfg) Configure the ADC with user configurations. More...
ssp_err_t	SF_AUDIO_RECORD_ADC_Close (sf_audio_record_ctrl_t *const p_api_ctrl)

	Call the ADC periodic framework Close. More...
ssp_err_t	SF_AUDIO_RECORD_ADC_Start (sf_audio_record_ctrl_t *const p_api_ctrl) Call the ADC periodic framework Start. More...
ssp_err_t	SF_AUDIO_RECORD_ADC_Stop (sf_audio_record_ctrl_t *const p_api_ctrl) Call the ADC periodic framework Stop. More...
ssp_err_t	SF_AUDIO_RECORD_ADC_InfoGet (sf_audio_record_ctrl_t *const p_api_ctrl, sf_audio_record_info_t *p_info) Provide information about the channel supported by audio recording framework(MONO) More...
ssp_err_t	SF_AUDIO_RECORD_ADC_VersionGet (ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

RTOS-integrated ADC implementation of Audio Recording Interface.

The Audio Recording Framework implementation uses the ADC periodic interface for audio recording.

Macro Definition Documentation

◆ SF_AUDIO_RECORD_CODE_VERSION_MAJOR

#define SF_AUDIO_RECORD_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ SF_AUDIO_RECORD_ADC_Close()

ssp_err_t SF_AUDIO_RECORD_ADC_Close (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Call the ADC periodic framework Close.

Implements

- sf_audio_record_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

Returns

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- sf_adc_periodic_api_t::close

Call the underlying ADC periodic close

◆ SF_AUDIO_RECORD_ADC_InfoGet()

ssp_err_t SF_AUDIO_RECORD_ADC_InfoGet (sf_audio_record_ctrl_t *const *p_api_ctrl*, sf_audio_record_info_t * *p_info*)

Provide information about the channel supported by audio recording framework(MONO)

Implements

- sf_audio_record_api_t::infoGet.

Return values

SSP_SUCCESS	InfoGet returns successfully.
SSP_ERR_ASSERTION	p_ctrl or p_info is null.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

Returns

See Common Error Codes or sf_adc_periodic for other possible return codes or causes.

Verify the parameters are valid

Set the channel support as MONO

◆ SF_AUDIO_RECORD_ADC_Open()

ssp_err_t SF_AUDIO_RECORD_ADC_Open (sf_audio_record_ctrl_t *const *p_api_ctrl*, sf_audio_record_cfg_t const *const *p_cfg*)

Configure the ADC with user configurations.

The SF_AUDIO_RECORD_ADC_Open will initialize the configurations for the underlying ADC periodic framework.

Implements

- sf_audio_record_api_t::open.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL. Or any one of the following p_cfg parameter is NULL/zero. p_cfg->p_capture_data_buffer, p_cfg->sample_count, p_cfg->capture_data_buffer_size, p_cfg->p_callback, p_cfg->sampling_rate_hz. Or resolution or time period is not matching. for other possible causes.
SSP_ERR_IN_USE	The channel specified has already been opened.

Returns

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- sf_adc_periodic_api_t::open

Note

This function is reentrant for any unit.

Initialize the ADC periodic framework

Initialize the configuration parameters for ADC periodic configuration structure

Configure the ADC resolution depending on the data width in cfg

Call the underlying ADC periodic open

◆ SF_AUDIO_RECORD_ADC_Start()

ssp_err_t SF_AUDIO_RECORD_ADC_Start (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Call the ADC periodic framework Start.

Implements

- sf_audio_record_api_t::start.

Return values

SSP_SUCCESS	ADC Periodic Scan started successfully.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

Returns

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- sf_adc_periodic_api_t::start

Call the underlying ADC periodic start

◆ SF_AUDIO_RECORD_ADC_Stop()

ssp_err_t SF_AUDIO_RECORD_ADC_Stop (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Call the ADC periodic framework Stop.

Implements

- sf_audio_record_api_t::stop.

Return values

SSP_SUCCESS	Periodic ADC scan stopped successfully.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

Returns

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- sf_adc_periodic_api_t::stop

Call the underlying ADC periodic start

◆ SF_AUDIO_RECORD_ADC_VersionGet()

ssp_err_t SF_AUDIO_RECORD_ADC_VersionGet (ssp_version_t *const *p_version*)
Gets version and stores it in provided pointer p_version. **Implements** ◦ sf_audio_record_api_t::versionGet. **Return values** SSP_SUCCESS — VersionGet returned successfully. SSP_ERR_ASSERTION — Parameter p_version was null.

sf_audio_record_adc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » ADC Audio recording Framework

Control block for audio recording Initialization occurs when sf_audio_record_api_t::open is called. More...

#include <sf_audio_record_adc.h>

Data Fields

uint32_t	open
TX_MUTEX	mutex
void *	p_capture_data_buffer
uint32_t	sample_count
void(*	p_callback)(sf_audio_record_callback_args_t *p_args) Callback function.
void const *	p_context Placeholder for user data.
sf_adc_periodic_instance_t const *	p_lower_lvl_adc_periodic Lower level ADC periodic instance.

Detailed Description

Control block for audio recording Initialization occurs when sf_audio_record_api_t::open is called.

Field Documentation

◆ mutex

TX_MUTEX sf_audio_record_adc_instance_ctrl_t::mutex
Mutex used to protect access to lower level driver hardware registers

◆ open

uint32_t sf_audio_record_adc_instance_ctrl_t::open
Used by driver to check if pointer to control block is valid

◆ p_capture_data_buffer

void* sf_audio_record_adc_instance_ctrl_t::p_capture_data_buffer
Pointer to the buffer that will store the samples

◆ sample_count

uint32_t sf_audio_record_adc_instance_ctrl_t::sample_count
Samples per channel to be buffered before notifying the app

The documentation for this struct was generated from the following file:

- sf_audio_record_adc.h

5.1.3.6 I2S Audio recording Framework

Renesas Synergy Software Package Reference » Framework Layer

I2S implementation of Audio Recording Interface. More...

Data Structures

struct	sf_audio_record_i2s_instance_ctrl_t

Macros

#define	SF_AUDIO_RECORD_I2S_CODE_VERSION_MAJOR (2U)
#define	SF_AUDIO_RECORD_I2S_OPEN (0x49325352)

Functions

ssp_err_t	SF_AUDIO_RECORD_I2S_Open (sf_audio_record_ctrl_t *const p_api_ctrl, sf_audio_record_cfg_t const *const p_cfg) Configures the Audio record I2S framework and I2S HAL driver The SF_AUDIO_RECORD_I2S_Open function creates a mutex for the I2S Unit used, then calls the driver open function. The mutex is deleted following the driver layer open function fails. More...
ssp_err_t	SF_AUDIO_RECORD_I2S_Start (sf_audio_record_ctrl_t *const p_api_ctrl) Gets mutex, starts recording data into the buffer. More...
ssp_err_t	SF_AUDIO_RECORD_I2S_Stop (sf_audio_record_ctrl_t *const p_api_ctrl) Stops the audio recording and releases mutex. More...
ssp_err_t	SF_AUDIO_RECORD_I2S_InfoGet (sf_audio_record_ctrl_t *const p_api_ctrl, sf_audio_record_info_t *p_info) Provide information about the channel supported by audio recording framework(stereo) More...
ssp_err_t	SF_AUDIO_RECORD_I2S_Close (sf_audio_record_ctrl_t *const p_api_ctrl) Closes I2S driver, releases and deletes the mutex. More...
ssp_err_t	SF_AUDIO_RECORD_I2S_VersionGet (ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

I2S implementation of Audio Recording Interface.

The Audio Recording Framework implementation uses the I2S interface for audio recording.

Macro Definition Documentation

◆ SF_AUDIO_RECORD_I2S_CODE_VERSION_MAJOR

#define SF_AUDIO_RECORD_I2S_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ SF_AUDIO_RECORD_I2S_OPEN

#define SF_AUDIO_RECORD_I2S_OPEN (0x49325352)
Macro definitions "I2SR" in ASCII

Function Documentation

◆ SF_AUDIO_RECORD_I2S_Close()

ssp_err_t SF_AUDIO_RECORD_I2S_Close (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Closes I2S driver, releases and deletes the mutex.

Implements

- sf_audio_record_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.

Returns

See Common Error Codes and lower level driver function for other possible return codes or causes. This function calls

- i2s_api_t::close

Call the underlying I2S close

Post the mutex

If all the HAL layers closed successfully, then delete the mutex and mark module as un-initialized

Clear information from control block so other functions know this instance is closed

◆ SF_AUDIO_RECORD_I2S_InfoGet()

ssp_err_t SF_AUDIO_RECORD_I2S_InfoGet (sf_audio_record_ctrl_t *const *p_api_ctrl*, sf_audio_record_info_t * *p_info*)

Provide information about the channel supported by audio recording framework(stereo)

Implements

- sf_audio_record_api_t::infoGet.

Return values

SSP_SUCCESS	InfoGet returns successfully.
SSP_ERR_ASSERTION	p_ctrl or p_info is null.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.

Returns

See Common Error Codes for other possible return codes or causes.

◆ SF_AUDIO_RECORD_I2S_Open()

ssp_err_t SF_AUDIO_RECORD_I2S_Open (sf_audio_record_ctrl_t *const *p_api_ctrl*, sf_audio_record_cfg_t const *const *p_cfg*)

Configures the Audio record I2S framework and I2S HAL driver The SF_AUDIO_RECORD_I2S_Open function creates a mutex for the I2S Unit used, then calls the driver open function. The mutex is deleted following the driver layer open function fails.

Implements

- sf_audio_record_api_t::open.

Return values

SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_cfg parameter is null:
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- i2s_api_t::open

Open I2S module

Create a mutex to protect access to the control structure and the lower level hardware.

Initialize the configuration parameters

If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code

Delete the mutex

Mark control block open so other tasks know it is valid

◆ SF_AUDIO_RECORD_I2S_Start()

ssp_err_t SF_AUDIO_RECORD_I2S_Start (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Gets mutex, starts recording data into the buffer.

Implements

- sf_audio_record_api_t::start.

Return values

SSP_SUCCESS	I2S record started successfully.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid.
SSP_ERR_IN_USE	The module is currently busy performing another operation

Returns

See Common Error Codes and lower level driver function for other possible return codes.
This function calls:

- i2s_api_t::read

Get mutex, start i2s

Call the underlying I2S driver read

If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code

Return the mutex

◆ SF_AUDIO_RECORD_I2S_Stop()

ssp_err_t SF_AUDIO_RECORD_I2S_Stop (sf_audio_record_ctrl_t *const *p_api_ctrl*)

Stops the audio recording and releases mutex.

Implements

- sf_audio_record_api_t::stop.

Return values

SSP_SUCCESS	I2S Record stopped successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.

Returns

See Common Error Codes for other possible return codes or causes. This function calls: i2s_api_t::stop

Return the mutex

If there was a mutex error, return it

◆ SF_AUDIO_RECORD_I2S_VersionGet()

ssp_err_t SF_AUDIO_RECORD_I2S_VersionGet (ssp_version_t *const *p_version*)

Gets version and stores it in provided pointer p_version.

Implements

- sf_audio_record_api_t::versionGet.

Return values

SSP_SUCCESS	VersionGet returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

Copy the version information.

sf_audio_record_i2s_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » I2S Audio recording Framework

#include <sf_audio_record_i2s.h>

Data Fields

uint32_t	open Used by driver to check if pointer to control.
TX_MUTEX	mutex Mutex used to protect access to lower level driver hardware registers.
void *	p_capture_data_buffer Pointer to the buffer record buffer */.
uint32_t	capture_data_size capture data type
uint32_t	data_size Number of bytes captured for each iteration.
uint32_t	buffer_size size of the current record buffer */
uint32_t	current_buffer_index Index into current buffer */.
void(*	p_callback)(sf_audio_record_callback_args_t *p_args) Callback function.
void const *	p_context Placeholder for user data.
i2s_instance_t const *	p_lower_lvl_i2s Lower level I2S instance.

Detailed Description

Control block for audio recording Initialization occurs when sf_audio_record_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_audio_record_i2s.h

5.1.3.7 BLOCK_MEDIA_LEVELX_NOR

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Block Media framework for LEVELX driver. More...

Data Structures

struct	sf_block_media_on_lx_nor_cfg_t
struct	sf_block_media_lx_nor_instance_ctrl_t

Macros

#define	SF_BLOCK_MEDIA_LX_NOR_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_block_media_lx_nor_version)
#define	SF_BLOCK_MEDIA_LX_NOR_OPEN (0x424D4C4FU)

Functions

ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_Open (sf_block_media_ctrl_t *const p_ctrl, sf_block_media_cfg_t const *const p_cfg) Open device for read/write and control. More...
ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_Read (sf_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count) Read data from flash using LevelX. More...
ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_Write (sf_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count) Write data to flash using LevelX. More...
ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_Control (sf_block_media_ctrl_t *const

	p_ctrl, ssp_command_t const command, void *p_data) Send control commands to Block Media LevelX NOR driver. More...
ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_Close (sf_block_media_ctrl_t *const p_ctrl) Close open Block Media LevelX NOR driver. More...
ssp_err_t	SF_BLOCK_MEDIA_LX_NOR_VersionGet (ssp_version_t *const p_version) Get version of Block Media LevelX driver. More...

Variables

const sf_block_media_api_t	g_sf_block_media_on_sf_block_media_lx_nor

Detailed Description

RTOS-integrated Block Media framework for LEVELX driver.

Macro Definition Documentation

◆ SF_BLOCK_MEDIA_LX_NOR_ERROR_RETURN

#define SF_BLOCK_MEDIA_LX_NOR_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_block_media_lx_nor_version)
Macro for error logger.

◆ SF_BLOCK_MEDIA_LX_NOR_OPEN

#define SF_BLOCK_MEDIA_LX_NOR_OPEN (0x424D4C4FU)
"BMLO" in ASCII, used to identify block media LevelX handle

Function Documentation

◆ SF_BLOCK_MEDIA_LX_NOR_Close()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_Close (sf_block_media_ctrl_t *const *p_ctrl*)

Close open Block Media LevelX NOR driver.

Close an open Block Media LevelX NOR driver.

Return values

SSP_SUCCESS	Successfully closed.
SSP_ERR_ASSERTION	p_ctrl or p_nor_flash is NULL.
SSP_ERR_NOT_OPEN	The block media is not open.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- lx_nor_flash_close
- SF_EL_LX_NOR_Close

Validate the parameters

Check whether the instance is in open state

Close the LevelX NOR flash driver.

Close underlying NOR driver, if available

Mark control block close so subsequent calls know the device is close.

◆ SF_BLOCK_MEDIA_LX_NOR_Control()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_Control (sf_block_media_ctrl_t *const *p_ctrl*, ssp_command_t const *command*, void * *p_data*)

Send control commands to Block Media LevelX NOR driver.

Return values

SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_data is Null.
SSP_ERR_NOT_OPEN	The block media is not open.
SSP_ERR_UNSUPPORTED	This module doesn't support requested command.
SSP_ERR_SECTOR_RELEASE_FAILED	Sector release command failed.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- lx_nor_flash_sector_release

Validate the parameters

Check whether the instance is in open state

Get the sector count

LevelX divides each NOR flash block into 512-byte logical sectors

It's not write protected

LevelX supports sector release

Release NOR flash sector.

◆ SF_BLOCK_MEDIA_LX_NOR_Open()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_Open (sf_block_media_ctrl_t *const *p_ctrl*, sf_block_media_cfg_t const *const *p_cfg*)

Open device for read/write and control.

Open LevelX flash device for read/write and control. This function initializes the LevelX driver and hardware the first time it is called out of reset. The underlying flash needs to either be erased or already initialized with LevelX.

Return values

SSP_SUCCESS	LevelX flash is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_ctrl, p_cfg or an input pointer is NULL.
SSP_ERR_MEDIA_OPEN_FAILED	LevelX NOR or the underlying flash failed to open. The underlying flash needs to either be erased or already initialized with LevelX.
SSP_ERR_ALREADY_OPEN	The block media LevelX NOR instance has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the instance.

Returns

See Common Error Codes or lower LevelX drivers for other possible return codes. This function calls:

- lx_nor_flash_open

Validate the parameters

Check whether instance is already open

Update instance control block

Open underlying LevelX

Mark control block open so subsequent calls know the device is open.

◆ SF_BLOCK_MEDIA_LX_NOR_Read()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_Read (sf_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Read data from flash using LevelX.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	p_ctrl or p_dest is NULL.
SSP_ERR_NOT_OPEN	The block media is not open.
SSP_ERR_READ_FAILED	Data read failed.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- lx_nor_flash_sector_read

Validate the parameters

Check whether the instance is in open state

Loop to read sectors from flash.

Read a sector from NOR flash.

◆ SF_BLOCK_MEDIA_LX_NOR_VersionGet()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_VersionGet (ssp_version_t *const *p_version*)

Get version of Block Media LevelX driver.

Return the version of the firmware and API.

Return values

SSP_ERR_ASSERTION	p_version is Pointer.
SSP_SUCCESS	version read successfully.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant.

Validate the parameters

◆ SF_BLOCK_MEDIA_LX_NOR_Write()

ssp_err_t SF_BLOCK_MEDIA_LX_NOR_Write (sf_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Write data to flash using LevelX.

Return values

SSP_SUCCESS	Write finished successfully.
SSP_ERR_ASSERTION	p_ctrl or p_src is NULL.
SSP_ERR_NOT_OPEN	The block media is not open.
SSP_ERR_WRITE_FAILED	Data write failed.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- lx_nor_flash_sector_write

Validate the parameters

Check whether the instance is in open state

Loop to write sectors into flash.

Write a sector into NOR flash.

Variable Documentation

◆ g_sf_block_media_on_sf_block_media_lx_nor

const sf_block_media_api_t g_sf_block_media_on_sf_block_media_lx_nor

```
=
{
    .open       = SF_BLOCK_MEDIA_LX_NOR_Open,
    .read       = SF_BLOCK_MEDIA_LX_NOR_Read,
    .write      = SF_BLOCK_MEDIA_LX_NOR_Write,
    .ioctl      = SF_BLOCK_MEDIA_LX_NOR_Control,
    .close      = SF_BLOCK_MEDIA_LX_NOR_Close,
    .versionGet = SF_BLOCK_MEDIA_LX_NOR_VersionGet
}
```

Block Media LevelX function pointers

sf_block_media_on_lx_nor_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLOCK_MEDIA_LEVELX_NOR

#include <sf_block_media_lx_nor.h>

Data Fields

UINT(*	nor_driver_initialize)(LX_NOR_FLASH *) Pointer to the initialization function.
LX_NOR_FLASH *	p_nor_flash NOR Flash instance.
CHAR *	p_nor_flash_name NOR Flash instance name.
ssp_err_t(*	close)() Pointer to underlying driver close.

Detailed Description

LevelX NOR block media config structure

The documentation for this struct was generated from the following file:

- sf_block_media_lx_nor.h

sf_block_media_lx_nor_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLOCK_MEDIA_LEVELX_NOR

#include <sf_block_media_lx_nor.h>

Data Fields

LX_NOR_FLASH *	p_nor_flash

	NOR Flash instance.
CHAR *	p_nor_flash_name NOR Flash instance name.
uint32_t	block_size Block size in bytes.
uint32_t	open Used to determine if framework is initialized.
ssp_err_t(*	close)() Pointer to underlying driver close.

Detailed Description

LevelX NOR block media instance control block.

The documentation for this struct was generated from the following file:

- sf_block_media_lx_nor.h

5.1.3.8 BLOCK_MEDIA_QSPI

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Block Media framework for QSPI driver. More...

Data Structures

struct	sf_block_media_qspi_instance_ctrl_t

Macros

#define	SF_QSPI_OPEN (0x51535049U)
#define	SF_BLOCK_MEDIA_QSPI_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_block_media_qspi_version)

Functions

ssp_err_t	SF_BLOCK_MEDIA_QSPI_Open (sf_block_media_ctrl_t *const p_api_ctrl, sf_block_media_cfg_t const *const p_cfg) Open Block Media QSPI flash device for read/write and control. Parameter checking and Acquires mutex, then handles driver initialization at the HAL QSPI layer and marking the open flag in control block. More...
ssp_err_t	SF_BLOCK_MEDIA_QSPI_Read (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const block_count) Read requested data from Block Media QSPI Flash through QSPI channel. More...
ssp_err_t	SF_BLOCK_MEDIA_QSPI_Write (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const block_count) Program requested data content to the Block Media QSPI flash memory. More...
ssp_err_t	SF_BLOCK_MEDIA_QSPI_Control (sf_block_media_ctrl_t *const p_api_ctrl, ssp_command_t const command, void *p_data) Send control commands to and receive status of flash. More...
ssp_err_t	SF_BLOCK_MEDIA_QSPI_Close (sf_block_media_ctrl_t *const p_api_ctrl) Close functionality will delete the resources which is initialized in open call. More...
ssp_err_t	SF_BLOCK_MEDIA_QSPI_VersionGet (ssp_version_t *const p_version) Get the firmware and API version of Block Media QSPI Framework. More...

Variables

const sf_block_media_api_t	g_sf_block_media_on_sf_block_media_qspi

Detailed Description

RTOS-integrated Block Media framework for QSPI driver.

Macro Definition Documentation

◆ SF_BLOCK_MEDIA_QSPI_ERROR_RETURN

#define SF_BLOCK_MEDIA_QSPI_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_block_media_qspi_version)
Macro for error logger.

◆ SF_QSPI_OPEN

#define SF_QSPI_OPEN (0x51535049U)
"QSPI" in ASCII, used to identify general SF_BLOCK_MEDIA_QSPI control block

Function Documentation

◆ SF_BLOCK_MEDIA_QSPI_Close()

ssp_err_t SF_BLOCK_MEDIA_QSPI_Close (sf_block_media_ctrl_t *const *p_api_ctrl*)

Close functionality will delete the resources which is initialized in open call.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- qspi_api_t::close

Delete RTOS services allocated during the open call.

◆ SF_BLOCK_MEDIA_QSPI_Control()

ssp_err_t SF_BLOCK_MEDIA_QSPI_Control (sf_block_media_ctrl_t *const *p_api_ctrl*, ssp_command_t const *command*, void * *p_data*)

Send control commands to and receive status of flash.

Return values

SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_UNSUPPORTED	Command not support.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- qspi_api_t::infoGet

Get the information of Flash

◆ SF_BLOCK_MEDIA_QSPI_Open()

ssp_err_t SF_BLOCK_MEDIA_QSPI_Open (sf_block_media_ctrl_t *const *p_api_ctrl*, sf_block_media_cfg_t const *const *p_cfg*)

Open Block Media QSPI flash device for read/write and control. Parameter checking and Acquires mutex, then handles driver initialization at the HAL QSPI layer and marking the open flag in control block.

Name of module used by error logger macro

Return values

SSP_SUCCESS	Block media for QSPI framework is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl, p_cfg or configuration for qspi.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- qspi_api_t::open

Create a mutex to protect access to the control structure and the lower level hardware.

Calling low level driver

On success populate the control structure

Selecting smallest block erasable by underlying QSPI flash

◆ SF_BLOCK_MEDIA_QSPI_Read()

ssp_err_t SF_BLOCK_MEDIA_QSPI_Read (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *block_count*)

Read requested data from Block Media QSPI Flash through QSPI channel.

Return values

SSP_SUCCESS	QSPI data read successfully.
SSP_ERR_ASSERTION	p_api_ctrl or p_dest is NULL. Or block_count is zero.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- qspi_api_t::read

Obtain mutex before making HAL-level driver call.

Release mutex

◆ SF_BLOCK_MEDIA_QSPI_VersionGet()

ssp_err_t SF_BLOCK_MEDIA_QSPI_VersionGet (ssp_version_t *const *p_version*)

Get the firmware and API version of Block Media QSPI Framework.

Return values

SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	Null Pointer.

Note

This function is reentrant.

◆ SF_BLOCK_MEDIA_QSPI_Write()

ssp_err_t SF_BLOCK_MEDIA_QSPI_Write (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *block_count*)

Program requested data content to the Block Media QSPI flash memory.

Return values

SSP_SUCCESS	Flash write finished successfully.
SSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL. Or block_count is zero.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_UNSUPPORTED	Total number of programmable bytes is greater than total flash capacity

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- qspi_api_t::pageProgram

Obtain mutex before making HAL-level driver call.

Get the information of Flash

Calculate the address of flash

Check if total number of bytes programmable into flash is greater than total flash capacity

Calculate the number of iteration required to complete write operation

Write data into the block media QSPI flash

Release mutex

Variable Documentation

◆ g_sf_block_media_on_sf_block_media_qspi

const sf_block_media_api_t g_sf_block_media_on_sf_block_media_qspi

```
=
{
        .open       = SF_BLOCK_MEDIA_QSPI_Open,
        .read       = SF_BLOCK_MEDIA_QSPI_Read,
        .write      = SF_BLOCK_MEDIA_QSPI_Write,
        .ioctl      = SF_BLOCK_MEDIA_QSPI_Control,
        .close      = SF_BLOCK_MEDIA_QSPI_Close,
        .versionGet = SF_BLOCK_MEDIA_QSPI_VersionGet
}
```

Block Media QSPI function pointers

sf_block_media_qspi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLOCK_MEDIA_QSPI

#include <sf_block_media_qspi.h>

Data Fields

uint32_t	open Used to determine if framework is initialized.
uint32_t	block_size Block size in bytes.
qspi_instance_t *	p_lower_lvl_qspi Pointer to QSPI instance structure.
TX_MUTEX	mutex Mutex used to protect access to lower level driver hardware registers.

Detailed Description

QSPI block media instance control block.

The documentation for this struct was generated from the following file:

- sf_block_media_qspi.h

5.1.3.9 BLOCK_MEDIA_RAM

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Block Media framework for RAM. More...

Data Structures

struct	sf_block_media_ram_instance_ctrl_t

Functions

ssp_err_t	SF_BLOCK_MEDIA_RAM_Open (sf_block_media_ctrl_t *const p_api_ctrl, sf_block_media_cfg_t const *const p_cfg) Open device for read/write and control. More...
ssp_err_t	SF_BLOCK_MEDIA_RAM_Read (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const block_count) Read data from RAM buffer. More...
ssp_err_t	SF_BLOCK_MEDIA_RAM_Write (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const block_count) Write data to RAM buffer. More...
ssp_err_t	SF_BLOCK_MEDIA_RAM_Control (sf_block_media_ctrl_t *const p_api_ctrl, ssp_command_t const command, void *p_data) Send control commands to and receive status of RAM buffer. More...
ssp_err_t	SF_BLOCK_MEDIA_RAM_Close (sf_block_media_ctrl_t *const p_api_ctrl) Close the Framework. More...

ssp_err_t	SF_BLOCK_MEDIA_RAM_VersionGet (ssp_version_t *const p_version) Get version of Block Media RAM framework. More...

Detailed Description

RTOS-integrated Block Media framework for RAM.

Function Documentation

◆ SF_BLOCK_MEDIA_RAM_Close()

ssp_err_t SF_BLOCK_MEDIA_RAM_Close (sf_block_media_ctrl_t *const *p_api_ctrl*)

Close the Framework.

Return values

SSP_SUCCESS	RAM buffer is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.

Check if Framework is NOT open

Mark Framework as close

◆ SF_BLOCK_MEDIA_RAM_Control()

ssp_err_t SF_BLOCK_MEDIA_RAM_Control (sf_block_media_ctrl_t *const *p_api_ctrl*, ssp_command_t const *command*, void * *p_data*)

Send control commands to and receive status of RAM buffer.

Return values

SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	p_api_ctrl or p_data is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.
SSP_ERR_UNSUPPORTED	Command not supported.

Check if Framework is NOT open

◆ SF_BLOCK_MEDIA_RAM_Open()

ssp_err_t SF_BLOCK_MEDIA_RAM_Open (sf_block_media_ctrl_t *const *p_api_ctrl*, sf_block_media_cfg_t const *const *p_cfg*)

Open device for read/write and control.

Return values

SSP_SUCCESS	RAM buffer is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_api_ctrl, p_cfg, p_cfg->p_extend, or p_block_media_cfg->p_ram_buffer is NULL.
SSP_ERR_IN_USE	Framework is already open by someone else

Check if Framework is already in USE

Copy the block size and RAM buffer memory address to control structure for further operation

Mark device as a open

◆ SF_BLOCK_MEDIA_RAM_Read()

ssp_err_t SF_BLOCK_MEDIA_RAM_Read (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *block_count*)

Read data from RAM buffer.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	p_api_ctrl, p_dest or internal control block element is NULL.
SSP_ERR_NOT_OPEN	The Framework is not opened.
SSP_ERR_INVALID_BLOCKS	Invalid block passed to API

Check if Framework is NOT open

Copy the content of p_ram_buffer into destination

◆ SF_BLOCK_MEDIA_RAM_VersionGet()

ssp_err_t SF_BLOCK_MEDIA_RAM_VersionGet (ssp_version_t *const *p_version*)

Get version of Block Media RAM framework.

Return the version of the firmware and API.

Return values

SSP_ERR_ASSERTION	p_version is NULL.
SSP_SUCCESS	version read successfully.

Note

This function is reentrant.

◆ SF_BLOCK_MEDIA_RAM_Write()

ssp_err_t SF_BLOCK_MEDIA_RAM_Write (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *block_count*)

Write data to RAM buffer.

Return values

SSP_SUCCESS	Data write successfully.
SSP_ERR_ASSERTION	p_api_ctrl, p_src or internal control block element is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.
SSP_ERR_INVALID_BLOCKS	Invalid block passed to API

Check if Framework is NOT open

Copy the content of source into p_ram_buffer

sf_block_media_ram_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLOCK_MEDIA_RAM

#include <sf_block_media_ram.h>

Data Fields

uint8_t *	p_ram_buffer pointer to RAM buffer address

uint32_t	block_size Block size in bytes.
uint32_t	open Used to determine if framework is initialized.
uint32_t	ram_buffer_size Size of RAM buffer.

Detailed Description

RAM block media instance control block.

The documentation for this struct was generated from the following file:

- sf_block_media_ram.h

5.1.3.10 BLOCK_MEDIA_SDMMC

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Block Media framework for SDMMC driver. More...

Data Structures

struct	sf_block_media_sdmmc_instance_ctrl_t

Functions

ssp_err_t	SF_Block_Media_SDMMC_Open (sf_block_media_ctrl_t *const p_api_ctrl, sf_block_media_cfg_t const *const p_cfg) Open device for read/write and control. More...
ssp_err_t	SF_Block_Media_SDMMC_Read (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const block_count) Read data from SD/MMC. More...

ssp_err_t	SF_Block_Media_SDMMC_Write (sf_block_media_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const block_count) Write data to SDMMC channel. More...
ssp_err_t	SF_Block_Media_SDMMC_Control (sf_block_media_ctrl_t *const p_api_ctrl, ssp_command_t const command, void *p_data) Send control commands to and receive status of SD/MMC port. More...
ssp_err_t	SF_Block_Media_SDMMC_Close (sf_block_media_ctrl_t *const p_api_ctrl) Close open device port. More...
ssp_err_t	SF_Block_Media_SDMMC_VersionGet (ssp_version_t *const p_version) Get version of Block Media SD/MMC driver. More...

Detailed Description

RTOS-integrated Block Media framework for SDMMC driver.

Function Documentation

◆ SF_Block_Media_SDMMC_Close()

ssp_err_t SF_Block_Media_SDMMC_Close (sf_block_media_ctrl_t *const *p_api_ctrl*)

Close open device port.

Close an open SD/MMC device port.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl or p_sdmmc is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Mark control block as unused so it can be reconfigured.

◆ SF_Block_Media_SDMMC_Control()

ssp_err_t SF_Block_Media_SDMMC_Control (sf_block_media_ctrl_t *const *p_api_ctrl*, ssp_command_t const *command*, void * *p_data*)

Send control commands to and receive status of SD/MMC port.

Send control commands to the SD/MMC port and receive the status of the SD/MMC port.

Return values

SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_sdmmc or p_data is Null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sdmmc_api_t::control

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

◆ SF_Block_Media_SDMMC_Open()

ssp_err_t SF_Block_Media_SDMMC_Open (sf_block_media_ctrl_t *const *p_api_ctrl*, sf_block_media_cfg_t const *const *p_cfg*)

Open device for read/write and control.

Open an SD or MMC device port for read/write and control. This function initializes the SDMMC driver and hardware the first time it is called out of reset.

Return values

SSP_SUCCESS	Port is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_ctrl, p_cfg, p_block_media_cfg or p_sdmmc is NULL.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sdmmc_api_t::open

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Create SDMMC event flag and put it into context

Mark the stream as open by initializing "BMSO" in its ASCII equivalent.

Cleanup before logging the error

◆ SF_Block_Media_SDMMC_Read()

ssp_err_t SF_Block_Media_SDMMC_Read (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *block_count*)

Read data from SD/MMC.

Read data from an SD or MMC device port.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	p_ctrl, p_sdmmc or p_dest is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_READ_FAILED	Data read failed.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sdmmc_api_t::read

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Wait until read operation is completed. Event is signaled in event flag object.

◆ SF_Block_Media_SDMMC_VersionGet()

ssp_err_t SF_Block_Media_SDMMC_VersionGet (ssp_version_t *const *p_version*)

Get version of Block Media SD/MMC driver.

Return the version of the firmware and API.

Return values

SSP_ERR_ASSERTION	p_version is Pointer.
SSP_SUCCESS	version read successfully.

Note

This function is reentrant.

◆ SF_Block_Media_SDMMC_Write()

ssp_err_t SF_Block_Media_SDMMC_Write (sf_block_media_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *block_count*)

Write data to SDMMC channel.

Return values

SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	p_ctrl, p_sdmmc or p_src is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_WRITE_FAILED	Data write failed.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sdmmc_api_t::write

Note

This function is reentrant for different channels.

Wait until write operation is completed. Event is signaled in event flag object.

sf_block_media_sdmmc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLOCK_MEDIA_SDMMC

#include <sf_block_media_sdmmc.h>

Data Fields

uint32_t	block_size Block size in bytes.
sdmmc_instance_t *	p_lower_lvl_sdmmc Pointer to SDMMC instance structure.
TX_EVENT_FLAGS_GROUP	eventflag Pointer to the event flag object for SDMMC data transfer.
uint32_t	open

	Used to determine if framework is initialized.

Detailed Description

SDMMC block media instance control block.

The documentation for this struct was generated from the following file:

- sf_block_media_sdmmc.h

5.1.3.11 Cellular NSAL Implementation on NetX

Renesas Synergy Software Package Reference » Framework Layer

Cellular NetX NSAL interface implementation header file. More...

Macros

#define	SF_CELR_NSALNX_CODE_VERSION_MAJOR (2U)
#define	SF_CELR_NSALNX_CODE_VERSION_MINOR (0U)

Functions

void	sf_cellular_nsal_netx_driver (NX_IP_DRIVER *driver_req_ptr, sf_cellular_instance_t const *p_cellular_instance, sf_cellular_nsal_cfg_t *p_cellular_nsal_cfg) NetX IP Driver entry function. More...
void	sf_cellular_nsal_ppp_send_byte (UCHAR byte, sf_cellular_instance_t const *p_celr_instance) PPP Send Byte Callback function call from PPP Stack. More...
void	sf_cellular_nsal_invalid_packet_handler (NX_PACKET *p_packet_ptr, sf_cellular_instance_t const *p_celr_instance) Cellular framework nsal invalid callback handler. More...

Detailed Description

Cellular NetX NSAL interface implementation header file.

Macro Definition Documentation

◆ SF_CELR_NSALNX_CODE_VERSION_MAJOR

#define SF_CELR_NSALNX_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file Major Version of code that implements the API defined in this file

◆ SF_CELR_NSALNX_CODE_VERSION_MINOR

#define SF_CELR_NSALNX_CODE_VERSION_MINOR (0U)
Minor Version of code that implements the API defined in this file

Function Documentation

◆ sf_cellular_nsal_invalid_packet_handler()

void sf_cellular_nsal_invalid_packet_handler (NX_PACKET * *p_packet_ptr*, sf_cellular_instance_t const * *p_celr_instance*)

Cellular framework nsal invalid callback handler.

Parameters

[in]	p_packet_ptr	Pointer to invalid PPP Packet received
[in]	p_celr_instance	Pointer to cellular framework instance

Cellular framework nsal invalid callback handler.

Parameters

[in]	p_packet_ptr	Pointer to invalid PPP Packet received
[in]	p_celr_instance	Pointer to cellular framework instance

variable to get Memory compare result

Check whether packet contains "NO CARRIER" string, we are ignoring first and last 2 bytes of packet which are Carriage Return and Line Feed.

check whether no carrier string present

Disconnect Network

Soft reset the module

Provision the Module using callback

Reestablish data connection

Restart PPP

Release the packet

◆ sf_cellular_nsal_netx_driver()

void sf_cellular_nsal_netx_driver (NX_IP_DRIVER * *driver_req_ptr*, sf_cellular_instance_t const * *p_cellular_instance*, sf_cellular_nsal_cfg_t * *p_cellular_nsal_cfg*)

NetX IP Driver entry function.

Parameters

[in]	driver_req_ptr	Pointer to NetX IP Driver
[in]	p_cellular_instance	Pointer to cellular framework instance
[in]	p_cellular_nsal_cfg	Pointer to cellular nsal configuration structure

NetX IP Driver entry function.

Parameters

[in]	driver_req_ptr	Pointer to NetX IP Driver
[in]	p_cellular_instance	Pointer to cellular framework instance
[in]	p_cellular_nsal_cfg	Pointer to cellular nsal configuration structure

Interface link is UP. Send the packet

◆ sf_cellular_nsal_ppp_send_byte()

void sf_cellular_nsal_ppp_send_byte (UCHAR *byte*, sf_cellular_instance_t const * *p_celr_instance*)

PPP Send Byte Callback function call from PPP Stack.

NSAL PPP Send bytes callback API

Parameters

[in]	byte	Byte to send
[in]	p_celr_instance	Pointer to cellular framework instance
[in]	byte	Byte to send
[in]	p_celr_instance	Pointer to cellular framework instance

Transmit byte

5.1.3.12 Telnet Communication Framework on sf_comms_telnet

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Communications Framework NetX telnet server implementation. More...

Data Structures

struct	sf_comms_telnet_instance_ctrl_t
struct	sf_comms_telnet_cfg_t

Macros

#define	SF_COMMS_TELNET_OPEN (0x434D544EU)

Functions

ssp_err_t	SF_COMMS_TELNET_Open (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_cfg_t const *const p_cfg) Initializes the Telnet server and other operating system resources. More...
ssp_err_t	SF_COMMS_TELNET_Close (sf_comms_ctrl_t *const p_api_ctrl) Disconnect Telnet server and clean up resources. More...
ssp_err_t	SF_COMMS_TELNET_Read (sf_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout) Read data from the Telnet comms connection. More...
ssp_err_t	SF_COMMS_TELNET_Write (sf_comms_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout) Write data to the Telnet comms connection. More...
ssp_err_t	SF_COMMS_TELNET_Lock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type, UINT timeout) Acquire lock type for the Telnet comms instance. More...
ssp_err_t	SF_COMMS_TELNET_Unlock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type) Release lock type for the Telnet comms instance. More...

ssp_err_t	SF_COMMS_TELNET_VersionGet (ssp_version_t *const p_version) Get driver version. More...

Detailed Description

RTOS-integrated Communications Framework NetX telnet server implementation.

Macro Definition Documentation

◆ SF_COMMS_TELNET_OPEN

#define SF_COMMS_TELNET_OPEN (0x434D544EU)
"CMTN" in ASCII, used to identify general timer handle

Function Documentation

◆ SF_COMMS_TELNET_Close()

ssp_err_t SF_COMMS_TELNET_Close (sf_comms_ctrl_t *const *p_api_ctrl*)

Disconnect Telnet server and clean up resources.

Return values

SSP_SUCCESS	Connection successfully closed
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL • p_ctrl->p_telnet_server is NULL • Telnet server field nx_telnet_server_id indicate this instance is not created already thus invlaid instance of Telnet server.
SSP_ERR_NOT_OPEN	Connection is not open

Note

- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.
- This function is reentrant.

Check if connection is open

Update connection record.

Return connection not open if no record found

Stop and release Telnet server and ThreadX resources for this connection

Mark this connection uninitialized.

◆ SF_COMMS_TELNET_Lock()

ssp_err_t SF_COMMS_TELNET_Lock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*, UINT *timeout*)

Acquire lock type for the Telnet comms instance.

Return values

SSP_SUCCESS	Acquired requested lock on given connection.
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Connection is not open.
SSP_ERR_TIMEOUT	Acquiring requested lock timed-out.

Note

- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.

Check if connection is open

Get both lock if requested

Else get the lock type requested

◆ SF_COMMS_TELNET_Open()

ssp_err_t SF_COMMS_TELNET_Open (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_cfg_t const *const *p_cfg*)

Initializes the Telnet server and other operating system resources.

Return values

SSP_SUCCESS	comms Telnet instance opened successfully.
SSP_ERR_IN_USE	comms Telnet maximum connection limit reached.
SSP_ERR_ALREADY_OPEN	comms Telnet instance already open.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_cfg is NULL • Pointer p_cfg->p_extend is NULL • Pointer p_cfg_extend->p_telnet_server is NULL • Pointer p_cfg_extend->p_ip is NULL • Pointer p_cfg_extend->p_stack is NULL • Pointer p_cfg_extend->stack_size is invalid i.e. zero
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

Note

- Do not modify sf_comms_ctrl_t structure returned by this API as it is for module's internal purpose.
- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.
- This function is reentrant.

Initialize and start Telnet server and ThreadX resources for this connection

Mark this connection initialized.

◆ SF_COMMS_TELNET_Read()

ssp_err_t SF_COMMS_TELNET_Read (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, UINT const *timeout*)

Read data from the Telnet comms connection.

Return values

SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL • Pointer p_dest is NULL • Invalid read length i.e. bytes value is zero
SSP_ERR_NOT_OPEN	Connection is not open
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Receive mutex get' timed out • 'Queue receive' timed out
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

Note

- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.
- This API is reentrant.

Check if connection is open

Check and wait for client to be connected with timeout.

Get read lock.

Get data from read queue.

Release read lock.

◆ SF_COMMS_TELNET_Unlock()

ssp_err_t SF_COMMS_TELNET_Unlock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*)

Release lock type for the Telnet comms instance.

Return values

SSP_SUCCESS	Released requested lock on given connection..
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Connection is not open.

Note

- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.

Check if connection is open

Release Lock(s) on this connection.

◆ SF_COMMS_TELNET_VersionGet()

ssp_err_t SF_COMMS_TELNET_VersionGet (ssp_version_t *const *p_version*)

Get driver version.

Return values

SSP_SUCCESS	Operation successful
SSP_ERR_ASSERTION	p_version pointer is NULL

Note

- This function is reentrant.

◆ SF_COMMS_TELNET_Write()

ssp_err_t SF_COMMS_TELNET_Write (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, UINT const *timeout*)

Write data to the Telnet comms connection.

Return values

SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL. • Pointer p_src is NULL. • Invalid write length i.e. bytes value is zero.
SSP_ERR_NOT_OPEN	Connection is not open
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Transmit mutex get' timed out
SSP_ERR_OUT_OF_MEMORY	Couldn't allocate pool memory for Telnet server
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

Note

- For Telnet server specific API details, refer Telnet server user manual.
- For NetX specific API details, refer NetX user manual.
- For ThreadX specific API details, refer ThreadX user manual.
- This function is reentrant.

Check if connection is open

Check and wait for client to be connected with timeout.

Get write lock.

Send data to the connected client.

Release write lock.

sf_comms_telnet_instance_ctrl_t Struct Reference
Renesas Synergy Software Package Reference » Framework Layer » Telnet Communication Framework on sf_comms_telnet

#include <sf_comms_telnet.h>

Data Fields

TX_EVENT_FLAGS_GROUP	available Flag to tell if this connection is available or not.
TX_QUEUE	queue Queue of received bytes.
void(*	p_disconnect_callback)(sf_comms_callback_args_t *p_args) User callback for Client disconnection.

Detailed Description

NetX Telnet server communications driver configuration

The documentation for this struct was generated from the following file:

- sf_comms_telnet.h

sf_comms_telnet_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Telnet Communication Framework on sf_comms_telnet

#include <sf_comms_telnet.h>

Data Fields

void(*	p_disconnect_callback)(sf_comms_callback_args_t *p_args) User callback for Client disconnection.

Detailed Description

NetX Telnet server device communications driver configuration

The documentation for this struct was generated from the following file:

- sf_comms_telnet.h

5.1.3.13 Console Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Console Framework. More...

Data Structures

struct	sf_console_instance_ctrl_t

Macros

#define	SF_CONSOLE_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_CONSOLE_Open (sf_console_ctrl_t *const p_api_ctrl, sf_console_cfg_t const *const p_cfg) Initialize console framework and open low-level communication driver. More...
ssp_err_t	SF_CONSOLE_Close (sf_console_ctrl_t *const p_api_ctrl) Close the communications driver. More...
ssp_err_t	SF_CONSOLE_Parse (sf_console_ctrl_t *const p_api_ctrl, sf_console_menu_t const *const p_menu, uint8_t const *const p_input, uint32_t const bytes) Looks for input string in menu, and calls callback function if found. More...
ssp_err_t	SF_CONSOLE_Prompt (sf_console_ctrl_t *const p_api_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout) Prompt the user to input a command. More...
ssp_err_t	SF_CONSOLE_Read (sf_console_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout) Reads data into the destination byte by byte and echos input to the console. More...

ssp_err_t	SF_CONSOLE_Write (sf_console_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const timeout) Write a NULL terminated string to the console. More...
ssp_err_t	SF_CONSOLE_ArgumentFind (uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data) Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers. More...
void	SF_CONSOLE_CallbackNextMenu (sf_console_callback_args_t *p_args) Callback provided to continue parsing the next menu down. More...
ssp_err_t	SF_CONSOLE_VersionGet (ssp_version_t *const p_version) Console version get function. More...

Detailed Description

RTOS-integrated Console Framework.

This is a ThreadX aware console framework implemented using the SSP communications framework.

Macro Definition Documentation

◆ SF_CONSOLE_CODE_VERSION_MAJOR

#define SF_CONSOLE_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ SF_CONSOLE_ArgumentFind()

ssp_err_t SF_CONSOLE_ArgumentFind (uint8_t const *const *p_arg*, uint8_t const *const *p_str*, int32_t *const *p_index*, int32_t *const *p_data*)

Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers.

Return values

SSP_SUCCESS	Argument found successfully
SSP_ERR_ASSERTION	p_arg or p_str is NULL
SSP_ERR_INTERNAL	String passed after parsing command is NULL_CODE.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Search for first letter match at beginning of word.

If the input string matches the input argument, store the index of the character following the argument in p_index and the data at that index in p_data. Then return.

◆ SF_CONSOLE_CallbackNextMenu()

void SF_CONSOLE_CallbackNextMenu (sf_console_callback_args_t * *p_args*)

Callback provided to continue parsing the next menu down.

Parameters

[in]	p_args	Pointer to callback arguments to use in the next menu.

Next level menu is passed in as the user context parameter

Update current menu.

Check to see if the next menu command was input in the remaining string.

◆ SF_CONSOLE_Close()

ssp_err_t SF_CONSOLE_Close (sf_console_ctrl_t *const *p_api_ctrl*)

Close the communications driver.

Return values

SSP_SUCCESS	Console successfully closed
SSP_ERR_ASSERTION	Pointer to control block is NULL

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Close UART driver

◆ SF_CONSOLE_Open()

ssp_err_t SF_CONSOLE_Open (sf_console_ctrl_t *const *p_api_ctrl*, sf_console_cfg_t const *const *p_cfg*)

Initialize console framework and open low-level communication driver.

Return values

SSP_SUCCESS	Console channel is successfully opened
SSP_ERR_ASSERTION	Parameter check failed for one of the following : -Pointer to the control block is NULL -Pointer to the config block is NULL -Pointer p_cfg->p_initial_menu is NULL -Pointer p_cfg->p_comms is NULL -Pointer p_cfg->p_comms->p_api is NULL -Pointer p_cfg->p_comms->p_api->open is NULL

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Open UART driver

Store echo configuration and initial menu in control block

Prompt for input autostart is true

◆ SF_CONSOLE_Parse()

ssp_err_t SF_CONSOLE_Parse (sf_console_ctrl_t *const *p_api_ctrl*, sf_console_menu_t const *const *p_menu*, uint8_t const *const *p_input*, uint32_t const *bytes*)

Looks for input string in menu, and calls callback function if found.

Return values

SSP_SUCCESS	Data parsed successfully, command found and callback called.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.
SSP_ERR_ASSERTION	One or more input parameter pointers are invalid.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Print help menu if help command is entered

Go back one menu if previous menu command is entered.

Go back to root menu if root menu command is entered.

Look for matching commands, call callback if command found.

◆ SF_CONSOLE_Prompt()

ssp_err_t SF_CONSOLE_Prompt (sf_console_ctrl_t *const *p_api_ctrl*, sf_console_menu_t const *const *p_menu*, UINT const *timeout*)

Prompt the user to input a command.

Return values

SSP_SUCCESS	Received valid command and called callback
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_UNSUPPORTED	Command not found in the current menu.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Update stored current menu pointer if a new pointer is specified.

Print menu name followed by ">" to prompt for user input.

Lock the console UART framework to reserve exclusive access until the command completes.

Note

Transmission is only locked while the menu name is printed and while the input command is non-zero in length. This allow debug messages to print from other threads while echo is off or no input command has been entered.

Wait for input

Parse input and call associated user callback.

Command is complete, so unlock comms reception.

◆ SF_CONSOLE_Read()

ssp_err_t SF_CONSOLE_Read (sf_console_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, uint32_t const *timeout*)

Reads data into the destination byte by byte and echos input to the console.

Return values

SSP_SUCCESS	Data read completed successfully
SSP_ERR_ASSERTION	Parameter check failed for one of the following : -Pointer p_dest is NULL -Pointer to the control block is NULL -Pointer p_ctrl->p_comms is NULL -Pointer p_ctrl->p_comms->p_api is NULL -Pointer p_ctrl->p_comms->p_api->lock is NULL -Pointer p_ctrl->p_comms->p_api->unlock is NULL

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Lock the communications framework reception until carriage return is received.

Read one byte at a time, checking for carriage returns, backspace, delete, and escape codes.

Unlock the communications framework reception

◆ SF_CONSOLE_VersionGet()

ssp_err_t SF_CONSOLE_VersionGet (ssp_version_t *const *p_version*)

Console version get function.

Parameters

[in]	p_version	Version information stored here.

Return values

SSP_SUCCESS	Version stored in provided pointer.
SSP_ERR_ASSERTION	p_version was null.

Set version pointer

◆ SF_CONSOLE_Write()

ssp_err_t SF_CONSOLE_Write (sf_console_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *timeout*)

Write a NULL terminated string to the console.

Return values

SSP_SUCCESS	Data write completed successfully
SSP_ERR_ASSERTION	Pointer to the control block is NULL
SSP_ERR_INVALID_SIZE	If length passed is zero or length passed is greater than 128U (max value).

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any channel.

Write null terminated string. Calculate the length. If it isn't longer than the maximum, write the entire string to the console.

sf_console_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Console Framework

#include <sf_console.h>

Data Fields

sf_console_menu_t const *	p_current_menu Current menu is stored here.
sf_comms_instance_t const *	p_comms Pointer to communications driver instance.
uint8_t	new_line Whether to echo input commands to transmitter.
bool	echo Whether to echo input commands to transmitter.

uint8_t	input [SF_CONSOLE_MAX_INPUT_LENGTH] Input buffer used to store user input.

Detailed Description

Console instance control block. DO NOT INITIALIZE. Initialization occurs when sf_console_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_console.h

5.1.3.14 SSP Crypto Common Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto Common Framework Module. More...

Data Structures

struct	sf_crypto_instance_ctrl_t

Macros

#define	SF_CRYPTO_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_CRYPTO_Open (sf_crypto_ctrl_t *const p_api_ctrl, sf_crypto_cfg_t const *const p_cfg) SSP Crypto Framework Common Open operation. More...
ssp_err_t	SF_CRYPTO_Close (sf_crypto_ctrl_t *const p_api_ctrl) SSP Crypto Framework Common Close operation. More...
ssp_err_t	SF_CRYPTO_Lock (sf_crypto_ctrl_t *const p_api_ctrl) Locks the module. This API is utilized for locking shared resources. More...
ssp_err_t	SF_CRYPTO_Unlock (sf_crypto_ctrl_t *const p_api_ctrl)

	Unlocks the module. This API is utilized for unlocking shared resources. More...
ssp_err_t	SF_CRYPTO_StatusGet (sf_crypto_ctrl_t *const p_api_ctrl, sf_crypto_state_t *p_status) Gets the Crypto Common Framework module status. More...
ssp_err_t	SF_CRYPTO_VersionGet (ssp_version_t *const p_version) Gets the version of Crypto Common Framework module. More...

Detailed Description

RTOS-integrated Crypto Common Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_CODE_VERSION_MAJOR

#define SF_CRYPTO_CODE_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework Common Module

Function Documentation

◆ SF_CRYPTO_Close()

ssp_err_t SF_CRYPTO_Close (sf_crypto_ctrl_t *const *p_api_ctrl*)

SSP Crypto Framework Common Close operation.

Parameters

[in,out]	p_api_ctrl	Pointer to a Crypto framework control block

Return values

SSP_SUCCESS	Module was successfully closed.
SSP_ERR_NOT_OPEN	Module has not opened.
SSP_ERR_ASSERTION	NULL pointer is passed as an input parameter.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
SSP_ERR_CRYPTO_HAL_ERROR	Unable to successfully close the Crypto Common HAL module.

Check if the module has been opened. If not, return error.

Close a lower level crypto driver.

◆ SF_CRYPTO_Lock()

ssp_err_t SF_CRYPTO_Lock (sf_crypto_ctrl_t *const *p_api_ctrl*)

Locks the module. This API is utilized for locking shared resources.

Parameters

[in,out]	p_api_ctrl	Pointer to a Crypto framework control block

Return values

SSP_SUCCESS	Module resources are successfully locked.
SSP_ERR_TIMEOUT	Unable to get ownership of the mutex within the specified time.
SSP_ERR_INTERNAL	Thread suspension was aborted. Critical error.
SSP_ERR_ASSERTION	NULL pointer is passed.as an input parameter.
SSP_ERR_NOT_OPEN	The module is not yet opened.

Check if the module has been opened. If not, return error.

Acquire the mutex.

◆ SF_CRYPTO_Open()

ssp_err_t SF_CRYPTO_Open (sf_crypto_ctrl_t *const *p_api_ctrl*, sf_crypto_cfg_t const *const *p_cfg*)

SSP Crypto Framework Common Open operation.

Parameters

[in,out]	p_api_ctrl	Pointer to a Crypto framework control block
[in]	p_cfg	Pointer to a Crypto framework configuration structure

Return values

SSP_SUCCESS	Crypto framework was successfully opened.
SSP_ERR_ASSERTION	NULL pointer is passed.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
SSP_ERR_CRYPTO_HAL_ERROR	Crypto HAL driver retuned an error.

Check if the module has been opened. If opened, return error.

Open a lower level crypto driver and ensure the engine is initialized here or already in other SCE supported stack.

◆ SF_CRYPTO_StatusGet()

ssp_err_t SF_CRYPTO_StatusGet (sf_crypto_ctrl_t *const *p_api_ctrl*, sf_crypto_state_t * *p_status*)

Gets the Crypto Common Framework module status.

Parameters

[in]	p_api_ctrl	Pointer to a Crypto framework control block
[out]	p_status	Memory location to store module status.

Return values

SSP_SUCCESS	Status returned successfully.
SSP_ERR_ASSERTION	The parameter p_status is NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	This common module is not yet opened.

Check if the module has a valid / known status else return error.

Return the module status.

◆ SF_CRYPTO_Unlock()

ssp_err_t SF_CRYPTO_Unlock (sf_crypto_ctrl_t *const *p_api_ctrl*)

Unlocks the module. This API is utilized for unlocking shared resources.

Parameters

[in,out]	p_api_ctrl	Pointer to a Crypto framework control block

Return values

SSP_SUCCESS	Module resources are successfully unlocked.
SSP_ERR_ASSERTION	NULL pointer is passed as an input parameter.
SSP_ERR_INTERNAL	Mutex is not owned by a caller thread.
SSP_ERR_NOT_OPEN	The module is not yet opened.

Check if the module has been opened. If not, return error.

Return the mutex.

◆ SF_CRYPTO_VersionGet()

ssp_err_t SF_CRYPTO_VersionGet (ssp_version_t *const *p_version*)

Gets the version of Crypto Common Framework module.

Parameters

[out]	p_version	Pointer to the memory to store the version information.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the module version.

sf_crypto_instance_ctrl_t Struct Reference
Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Common Framework

#include <sf_crypto.h>

Data Fields

sf_crypto_state_t	status Module status.
TX_MUTEX	mutex Mutex used in the Crypto Framework.
TX_SEMAPHORE	semaphore Semaphore used in the Crypto Framework (Reserve)
TX_BYTE_POOL	byte_pool Byte pool used in the Crypto Framework.
uint32_t	wait_option Wait time option used for RTOS service calls.
uint32_t	open_counter Counter to keep the number of SF_CRYPTO_XXX opened.
void *	p_lower_lvl_crypto Pointer to a low-level Crypto engine HAL driver instance.
void(*	p_callback)(sf_crypto_callback_args_t *p_args) Pointer to callback function.
void *	p_context Pointer to a context.
sf_crypto_close_option_t	close_option Close option.

Detailed Description

SSP Crypto Framework Common Module instance control block

The documentation for this struct was generated from the following file:

- sf_crypto.h

5.1.3.15 SSP Crypto Cipher Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto Cipher Framework Module. More...

Data Structures

struct	sf_crypto_cipher_instance_ctrl_t

Macros

#define	SF_CRYPTO_CIPHER_CODE_VERSION_MAJOR (2U)
#define	SF_CRYPTO_CIPHER_API_VERSION_MAJOR (1U)
#define	SF_CRYPTO_CIPHER_AES_128_XTS_KEY_SIZE (2 * (AES128_SECRET_KEY_SIZE_BYTES))
#define	SF_CRYPTO_CIPHER_AES_GCM_TAG_LENGTH_16_BYTES (16U) AES GCM tag of length 16 bytes.
#define	SF_CRYPTO_CIPHER_AES_IV_LENGTH_12_BYTES (12U) IV for AES operations - 16 bytes.
#define	SF_CRYPTO_CIPHER_AES_GCM_IV_PAD_4_BYTES (4U) 4 byte padding for 96-bit IV.
#define	SF_CRYPTO_CIPHER_AES_IV_LENGTH_16_BYTES (16U) IV for AES operations - 16 bytes.
#define	SF_CRYPTO_CIPHER_AES_BLOCK_SIZE_BYTES (16U) AES block size = 16 bytes.

#define	SF_CRYPTO_CIPHER_BYTES_PER_WORD (4U) number of bytes in a WORD.
#define	SF_CRYPTO_CIPHER_RSA_1024_MODULUS_BITS (1024U) Modulus size of RSA 1024-bit key. More...
#define	SF_CRYPTO_CIPHER_RSA_2048_MODULUS_BITS (2048U) Modulus size of RSA 2048-bit key.
#define	SF_CRYPTO_PKCS_1_5_EB_START_BYTE (0U) Encryption Block start byte = 00.
#define	SF_CRYPTO_PKCS_1_5_BT_00 (0U) Encryption Block Type (BT) = 00.
#define	SF_CRYPTO_PKCS_1_5_BT_01 (1U) Encryption Block Type (BT) = 01.
#define	SF_CRYPTO_PKCS_1_5_BT_02 (2U) Encryption Block Type (BT) = 02.
#define	SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR (0U) EB Data separator (between PS and Data).
#define	SF_CRYPTO_PKCS_1_5_EB_START_BYTE_LENGTH (1U) Encryption Block Start Byte length.
#define	SF_CRYPTO_PKCS_1_5_EB_BT_BYTE_LENGTH (1U) Encryption Block Block Type field length.
#define	SF_CRYPTO_PKCS_1_5_EB_PS_MIN_LENGTH (8U) Encryption Block Padding String (PS) min length.

#define	SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR_LENGTH (1U) EB Data separator (between PS and Data)length.
#define	SF_CRYPTO_PKCS_1_5_EB_OVERHEAD

Enumerations

enum	sf_crypto_cipher_state_t {
	SF_CRYPTO_CIPHER_STATE_CLOSED,
	SF_CRYPTO_CIPHER_STATE_OPENED,
	SF_CRYPTO_CIPHER_STATE_INITIALIZED,
	SF_CRYPTO_CIPHER_STATE_UPDATED,
	SF_CRYPTO_CIPHER_STATE_FINALIZED
	}

Functions

void	sf_crypto_cipher_aes_interface_get (sf_crypto_cipher_instance_ctrl_t *const p_ctrl) Subroutine to get a pointer to the AES HAL API instance. More...
bool	sf_crypto_cipher_is_key_type_aes (sf_crypto_key_type_t key_type) Subroutine to check if the key type enum provided is a valid AES key type for cipher operation. More...
ssp_err_t	sf_crypto_cipher_aes_init (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const p_key, sf_crypto_cipher_algorithm_init_params_t *p_algorithm_specific_params) Subroutine to initialize the AES cipher operation. More...
ssp_err_t	sf_crypto_cipher_aes_update (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Subroutine to update the cipher AES operation. More...
ssp_err_t	sf_crypto_cipher_aes_validate_aad_update_params_context (sf_crypto_cipher_instance_ctrl_t *const p_ctrl) Subroutine to validate the context buffer parameters for the cipherAadUpdate operation. More...
ssp_err_t	sf_crypto_cipher_aes_aad_update (sf_crypto_cipher_instance_ctrl_t *p_ctrl, sf_crypto_data_handle_t const *const p_aad)

	Subroutine to update the AAD for the cipher operation. More...
ssp_err_t	sf_crypto_cipher_aes_encrypt_final (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Subroutine to finalize the cipher encrypt operation. More...
ssp_err_t	sf_crypto_cipher_aes_decrypt_final (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Subroutine to finalize the cipher decrypt operation. More...
ssp_err_t	sf_crypto_cipher_aes_final (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Sub-routine for Crypto Cipher Framework to call the appropriate routine for final encryption based on the algorithm for the cipherFinal operation. This function is called by SF_CRYPTO_CIPHER_CipherFinal(). More...
ssp_err_t	sf_crypto_cipher_initialize_aes_instance (sf_crypto_cipher_instance_ctrl_t *p_ctrl, sf_crypto_cipher_cfg_t const *const p_cfg) Allocates memory for the instance and opens the underlying HAL instance. More...
ssp_err_t	sf_crypto_cipher_deinitialize_aes_instance (sf_crypto_cipher_instance_ctrl_t *p_ctrl) closes the underlying HAL instance and releases the memory for the instance. More...
__STATIC_INLINE uint32_t	SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES (uint32_t key_size_bits) Routine to calculate and return the Encryption block size based on the key size. More...
__STATIC_INLINE uint32_t	SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES (uint32_t key_size_bits) Routine to calculate and return the Encryption block size based on the key size. More...

void	sf_crypto_cipher_rsa_interface_get (sf_crypto_cipher_instance_ctrl_t *const p_ctrl) Subroutine to get a RSA HAL API instance. This function is called by sf_crypto_cipher_open_rsa(). More...
bool	sf_crypto_cipher_is_key_type_rsa (sf_crypto_key_type_t key_type) Subroutine to check if key type is RSA. More...
ssp_err_t	sf_crypto_cipher_rsa_init (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const p_key, sf_crypto_cipher_algorithm_init_params_t *p_algorithm_specific_params) Routine to handle the cipherInit RSA operation. More...
ssp_err_t	sf_crypto_cipher_rsa_update (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Subroutine to update the cipher RSA operation. This algorithm is only suitable for messages of limited length. The total number of input bytes processed during encryption may not be more than k-11, where k is the RSA key's modulus size in bytes. The encryption block(EB) during encryption with a Public key is built as follows: EB = 00 || 02 || PS || 00 || M :: M (input bytes) is the plaintext message :: PS is an octet string of length k-3-||M|| of pseudo random nonzero octets. The length of PS must be at least 8 octets. :: k is the RSA modulus size. More...
ssp_err_t	sf_crypto_cipher_rsa_final (sf_crypto_cipher_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) Routine to process the final encryption / decryption operation for RSA. This function is called by SF_CRYPTO_CIPHER_CipherFinal(). More...
ssp_err_t	sf_crypto_cipher_initialize_rsa_instance (sf_crypto_cipher_instance_ctrl_t *p_ctrl, sf_crypto_cipher_cfg_t const *const p_cfg) Allocates memory for the instance and opens the underlying HAL instance. More...
ssp_err_t	sf_crypto_cipher_deinitialize_rsa_instance (sf_crypto_cipher_instance_ctrl_t *p_ctrl)

	closes the underlying HAL instance and releases the memory for the instance. More...
ssp_err_t	SF_CRYPTO_CIPHER_Open (sf_crypto_cipher_ctrl_t *const p_api_ctrl, sf_crypto_cipher_cfg_t const *const p_cfg) SSP Crypto Cipher Framework Open operation. More...
ssp_err_t	SF_CRYPTO_CIPHER_Close (sf_crypto_cipher_ctrl_t *const p_api_ctrl) SSP Crypto Cipher Framework Close operation. More...
ssp_err_t	SF_CRYPTO_CIPHER_VersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. More...
ssp_err_t	SF_CRYPTO_CIPHER_CipherInit (sf_crypto_cipher_ctrl_t *const p_api_ctrl, sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const p_key, sf_crypto_cipher_algorithm_init_params_t *const p_algorithm_specific_params) SSP Crypto Framework Cipher Init operation. The input parameters initialize the cipher operation. Some of the parameters are algorithm specific. The input parameters are validated and then copied into the context buffer. Refer to sf_crypto_cipher_aes_init_params_t or sf_crypto_cipher_rsa_init_params_t. More...
ssp_err_t	SF_CRYPTO_CIPHER_CipherUpdate (sf_crypto_cipher_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) SSP Crypto Cipher Framework - Encrypts / Decrypts the input data and writes the cipher-text or plain-text to the output buffer. Can be called multiple times for chunks of data. More...
ssp_err_t	SF_CRYPTO_CIPHER_CipherAadUpdate (sf_crypto_cipher_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_aad) Updates Additional Authenticated Data. This is applicable only to AES GCM. Can be called multiple times for chunks of data. More...
ssp_err_t	SF_CRYPTO_CIPHER_CipherFinal (sf_crypto_cipher_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const p_data_out) SSP Crypto Cipher Framework - Generates encrypted / decrypted output from all/last input data. This function processes any remaining input data buffered by one or more calls to the

	cipherUpdate()API as well as input data supplied in the input parameter. This function must be invoked to complete a cipher operation. More...

Detailed Description

RTOS-integrated Crypto Cipher Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_CIPHER_AES_128_XTS_KEY_SIZE

#define SF_CRYPTO_CIPHER_AES_128_XTS_KEY_SIZE (2 * (AES128_SECRET_KEY_SIZE_BYTES))
Macros for AES operations

◆ SF_CRYPTO_CIPHER_API_VERSION_MAJOR

#define SF_CRYPTO_CIPHER_API_VERSION_MAJOR (1U)
The API version of SSP Crypto CIPHER Framework

◆ SF_CRYPTO_CIPHER_CODE_VERSION_MAJOR

#define SF_CRYPTO_CIPHER_CODE_VERSION_MAJOR (2U)
The API version of SSP Crypto Cipher Framework

◆ SF_CRYPTO_CIPHER_RSA_1024_MODULUS_BITS

#define SF_CRYPTO_CIPHER_RSA_1024_MODULUS_BITS (1024U)
Modulus size of RSA 1024-bit key. Macros for RSA operations EB = ENCRYPTION_BLOCK PS = Padding String PKCS_1_5 = RSAES-PKCS1-v1_5

◆ SF_CRYPTO_PKCS_1_5_EB_OVERHEAD

#define SF_CRYPTO_PKCS_1_5_EB_OVERHEAD

```
(SF_CRYPTO_PKCS_1_5_EB_START_BYTE_LENGTH + \
                                        SF_CRYPTO_PKCS_1_5_EB_BT_BYTE_LENGTH +    \
                                        SF_CRYPTO_PKCS_1_5_EB_PS_MIN_LENGTH +    \
                                        SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR_LENGTH)
```

Overhead for formatting the Encryption Block, in number of bytes

Enumeration Type Documentation

◆ sf_crypto_cipher_state_t

enum sf_crypto_cipher_state_t

States the SSP Crypto Cipher Framework module can go through.

Enumerator	
SF_CRYPTO_CIPHER_STATE_CLOSED	The Cipher module is closed.
SF_CRYPTO_CIPHER_STATE_OPENED	The Cipher module is opened.
SF_CRYPTO_CIPHER_STATE_INITIALIZED	The cipher operation is initialized.
SF_CRYPTO_CIPHER_STATE_UPDATED	The cipher operation is updated.
SF_CRYPTO_CIPHER_STATE_FINALIZED	The cipher operation is finalized.

Function Documentation

◆ sf_crypto_cipher_aes_aad_update()

ssp_err_t sf_crypto_cipher_aes_aad_update (sf_crypto_cipher_instance_ctrl_t * *p_ctrl*, sf_crypto_data_handle_t const *const *p_aad*)

Subroutine to update the AAD for the cipher operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in]	p_aad	Pointer to the input data structure - has the pointer to input AAD and the data length

Return values

SSP_SUCCESS	AAD was updated successfully.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Check if there is data in the aad block buffer. If so first fill it from the input data

Update the number of bytes remaining in the input buffer to be processed.

If data is block length, encrypt it.

Check if the input data is a multiple of block size

Update the number of bytes remaining in the input buffer to be processed.

If any bytes remain that is less than the block size, copy to the partial buffer

Update the size of the data in the partial block

◆ sf_crypto_cipher_aes_decrypt_final()

ssp_err_t sf_crypto_cipher_aes_decrypt_final (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Subroutine to finalize the cipher decrypt operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in,out]	p_data_in	Pointer to the input data structure - has the pointer to input data and the data length
[in,out]	p_data_out	Pointer to the output data structure - has the pointer to output data and the data length.

Return values

SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Validate the input parameters

Process the input data

If GCM check if there is any partial AAD remaining, if so call to update AAD

If GCM set the tag

For GCM compute and verify the tag - return error to indicate if the data is valid or not.

◆ sf_crypto_cipher_aes_encrypt_final()

ssp_err_t sf_crypto_cipher_aes_encrypt_final (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Subroutine to finalize the cipher encrypt operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in,out]	p_data_in	Pointer to the input data structure - has the pointer to input data and the data length
[in,out]	p_data_out	Pointer to the output data structure - has the pointer to output data and the data length.

Return values

SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Validate the input parameters

Process the input data

Handle padding for ECB, CBC modes

Add padding and encrypt the last block.

If there is a remaining partial block process it. For GCM zero padding is automatic

Now that the possible AAD update and plain text encryption is done, get the GCM tag

◆ sf_crypto_cipher_aes_final()

ssp_err_t sf_crypto_cipher_aes_final (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Sub-routine for Crypto Cipher Framework to call the appropriate routine for final encryption based on the algorithm for the cipherFinal operation. This function is called by SF_CRYPTO_CIPHER_CipherFinal().

Parameters

[in,out]	p_ctrl	Pointer to the Cipher framework module instance control block.
[in]	p_data_in	Pointer to the input data buffer and the length of the input data.
[in,out]	p_data_out	Pointer to the output data buffer and the buffer length on input. If data is output, the length of the data will be updated.

Return values

SSP_SUCCESS	The cipher update operation was successful.
SSP_ERR_UNSUPPORTED	The module does not support the algorithm bbased on the key type specified by the user.

Returns

See Common Error Codes for other possible return codes.

Validate that there is some input data provided / processed before the finalizing operation Applies to both encrypt and decrypt operations. AES GCM is the only exception.

◆ sf_crypto_cipher_aes_init()

ssp_err_t sf_crypto_cipher_aes_init (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_cipher_op_mode_t *cipher_operation_mode*, sf_crypto_key_t const *const *p_key*, sf_crypto_cipher_algorithm_init_params_t * *p_algorithm_specific_params*)

Subroutine to initialize the AES cipher operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in]	cipher_operation_mode	The cipher operation mode - encrypt / decrypt.
[in]	p_key	Pointer to the key to be used for the cipher operations.
[in]	p_algorithm_specific_params	Pointer to the algorithm specific parameters.

Return values

SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

All buffers within the context buffer are allocated at Open. Now just zeroise their contents.

Copy the init parameters to the context buffer

◆ sf_crypto_cipher_aes_interface_get()

void sf_crypto_cipher_aes_interface_get (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*)

Subroutine to get a pointer to the AES HAL API instance.

Parameters

[in,out]	p_ctrl	Pointer to a Cipher framework control block, whose p_hal_api is filled with HAL AES interface. This will be NULL, for MCUs /feature/ configuration parameters not supported

Get a Crypto common control block and the HAL instance.

Check the AES key type and size and get an appropriate API instance.

Get the HAL API instance for a selected algorithm type.

◆ sf_crypto_cipher_aes_update()

ssp_err_t sf_crypto_cipher_aes_update (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Subroutine to update the cipher AES operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in,out]	p_data_in	Pointer to the input data structure - has the pointer to input data and the data length
[in,out]	p_data_out	Pointer to the output data structure - has the pointer to output data and the data length.

Return values

SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

If GCM check if there is any partial AAD remaining. This is to be processed only if the cipher is in initialized state. Once the update is done AAD calculations should not be done. if so call to update AAD

Set the data out length before starting cipher operation.

Update the number of bytes remaining in the input buffer to be processed.

If data is block length, process it. That is encrypt / decrypt it.

Check if the remaining input data is a multiple of block size

Update the number of bytes written to the output buffer.

Update the number of bytes remaining in the input buffer to be processed.

If any bytes remain that is less than the block size, copy to the partial buffer

Update the size of the data in the partial block

◆ sf_crypto_cipher_aes_validate_aad_update_params_context()

ssp_err_t sf_crypto_cipher_aes_validate_aad_update_params_context (
sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*)

Subroutine to validate the context buffer parameters for the cipherAadUpdate operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.

Return values

SSP_SUCCESS	Parameters were validated successfully.
SSP_ERR_ASSERTION	At least one of the buffer is set to NULL.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

◆ SF_CRYPTO_CIPHER_CipherAadUpdate()

ssp_err_t SF_CRYPTO_CIPHER_CipherAadUpdate (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_aad*)

Updates Additional Authenticated Data. This is applicable only to AES GCM. Can be called multiple times for chunks of data.

Return values

SSP_SUCCESS	The function updated the AAD successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED.

Returns

See Common Error Codes for other possible return codes.

Note

1. *This is a blocking call.*
2. *This function has to be called before any call to cipherUpdate / cipherFinal is made.*
3. *The data buffer must be WORD aligned.*

do nothing - similar to the cipherUpdate API

Get a Crypto Framework common control block and the interface.

Check if the Crypto Cipher Framework is in the correct state to execute this operation. If module is not opened or initialized, return an error.

Acquire the lock from the common module to access the Crypto HAL driver.

AAD is optional so no need to track if it is updated.

Unlock the module.

if update processing has succeeded return the error from unlock.

◆ SF_CRYPTO_CIPHER_CipherFinal()

ssp_err_t SF_CRYPTO_CIPHER_CipherFinal (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

SSP Crypto Cipher Framework - Generates encrypted / decrypted output from all/last input data. This function processes any remaining input data buffered by one or more calls to the cipherUpdate()API as well as input data supplied in the input parameter. This function must be invoked to complete a cipher operation.

Return values

SSP_SUCCESS	The module updated a message Digest successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED or SF_CRYPTO_CIPHER_STATE_UPDATED.

Returns

See Common Error Codes for other possible return codes.

Note

1. This is a blocking call.
2. On encryption and decryption operations, block alignment considerations may require that the number of bytes written into output buffer be larger or smaller than input data length or even 0.
3. The output data area must not partially overlap the input data area such that the input data is modified before it is used else incorrect output may result.
4. Except for GCM, if the length of input data is 0 and no input was provided through update, an error is returned.
5. On decryption operations the padding bytes are not written to p_data_out.p_data.
6. The input and output buffers must be WORD aligned.

Get a Crypto Framework common control block and the interface.

Check if the Crypto Cipher Framework is in the correct state to execute this operation. If module is not opened or initialized, return an error.

Acquire the lock from the common module to access to Crypto HAL driver.

Mark the module status as 'Finalized'.

Unlock the module.

if final processing has succeeded return the error from unlock.

◆ SF_CRYPTO_CIPHER_CipherInit()

ssp_err_t SF_CRYPTO_CIPHER_CipherInit (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*, sf_crypto_cipher_op_mode_t *cipher_operation_mode*, sf_crypto_key_t const *const *p_key*, sf_crypto_cipher_algorithm_init_params_t *const *p_algorithm_specific_params*)

SSP Crypto Framework Cipher Init operation. The input parameters initialize the cipher operation. Some of the parameters are algorithm specific. The input parameters are validated and then copied into the context buffer. Refer to sf_crypto_cipher_aes_init_params_t or sf_crypto_cipher_rsa_init_params_t.

Return values

SSP_SUCCESS	The module is initialized for cipher operation successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_INVALID_ARGUMENT	One of the input parameters is invalid.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.

Returns

See Common Error Codes for other possible return codes.

Validate cipher operation mode

Check if the module can transition to the initialized state.

Note: sf_crypto.lock() is not called because the HAL driver does not access HW during init. If HW will be accessed through the HAL driver, lock should be acquired.

Do algorithm specific init

Mark the module status as 'Initialized'.

◆ SF_CRYPTO_CIPHER_CipherUpdate()

ssp_err_t SF_CRYPTO_CIPHER_CipherUpdate (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

SSP Crypto Cipher Framework - Encrypts / Decrypts the input data and writes the cipher-text or plain-text to the output buffer. Can be called multiple times for chunks of data.

Return values

SSP_SUCCESS	The function updated a cipher operation successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED.

Returns

See Common Error Codes for other possible return codes.

Note

1. This is a blocking call.
2. The input and output buffers have to be WORD aligned.
3. On encryption and decryption operations, block alignment considerations may require that the number of bytes written into output buffer be larger or smaller than input data length or even 0.
4. The output data area must not partially overlap the input data area such that the input data is modified before it is used else incorrect output may result.
5. If the length of input data is 0 this method does nothing.
6. For all Cipher operations cipherFinal() must be called to finalize the operation.
7. For RSA Operations, no data is output unless cipherFinal() is called.

If input data length is 0 - do nothing return success.

The status will be set to 'updated' only when data is processed.

Get a Crypto Framework common control block and the interface.

Check if the Crypto Cipher Framework is in the correct state to transition to do the update operation. If module is not opened or initialized, return an error.

Acquire the lock from the common module to access the Crypto HAL driver.

If update processing has succeeded return the error from unlock.

◆ SF_CRYPTO_CIPHER_Close()

ssp_err_t SF_CRYPTO_CIPHER_Close (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*)

SSP Crypto Cipher Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	One or more input parameters maybe NULL.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto common control block and the interface.

Check if the Crypto Framework Cipher module has been opened. If not yet opened, return an error.

Note: sf_crypto.lock() is not called because the HAL driver does not access HW during close. If HW will be accessed through the HAL driver, lock should be acquired.

Free the memory allocated for this instance and close the HAL driver.

◆ sf_crypto_cipher_deinitialize_aes_instance()

ssp_err_t sf_crypto_cipher_deinitialize_aes_instance (sf_crypto_cipher_instance_ctrl_t * *p_ctrl*)

closes the underlying HAL instance and releases the memory for the instance.

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework instance control block structure.

Return values

SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

◆ sf_crypto_cipher_deinitialize_rsa_instance()

ssp_err_t sf_crypto_cipher_deinitialize_rsa_instance (sf_crypto_cipher_instance_ctrl_t * *p_ctrl*)

closes the underlying HAL instance and releases the memory for the instance.

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework instance control block structure.

Return values

SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

First close the Framework TRNG instance.

Then close the HAL RSA instance.

◆ sf_crypto_cipher_initialize_aes_instance()

ssp_err_t sf_crypto_cipher_initialize_aes_instance (sf_crypto_cipher_instance_ctrl_t * *p_ctrl*, sf_crypto_cipher_cfg_t const *const *p_cfg*)

Allocates memory for the instance and opens the underlying HAL instance.

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework instance control block structure.
[in]	p_cfg	Pointer to the configuration structure for Cipher module .

Return values

SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

◆ sf_crypto_cipher_initialize_rsa_instance()

ssp_err_t sf_crypto_cipher_initialize_rsa_instance (sf_crypto_cipher_instance_ctrl_t * *p_ctrl*, sf_crypto_cipher_cfg_t const *const *p_cfg*)

Allocates memory for the instance and opens the underlying HAL instance.

Parameters

[in,out]	p_ctrl	Pointer to Crypto Cipher Framework instance control block structure.
[in]	p_cfg	Pointer to the configuration structure for Cipher module .

Return values

SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

◆ sf_crypto_cipher_is_key_type_aes()

bool sf_crypto_cipher_is_key_type_aes (sf_crypto_key_type_t *key_type*)

Subroutine to check if the key type enum provided is a valid AES key type for cipher operation.

Parameters

[in]	key_type	The key type to be tested.

Return values

true	The key is a valid AES key type.
false	The key is NOT a valid AES key type.

Check the key type and and determine the algorithm.

◆ sf_crypto_cipher_is_key_type_rsa()

bool sf_crypto_cipher_is_key_type_rsa (sf_crypto_key_type_t *key_type*)

Subroutine to check if key type is RSA.

Parameters

[in]	key_type	Key type enum.

Return values

true	Key type is RSA plain text (standard or CRT)/ wrapped.
false	Key type is not RSA.

Check the key type is valid for RSA operations.

◆ SF_CRYPTO_CIPHER_Open()

ssp_err_t SF_CRYPTO_CIPHER_Open (sf_crypto_cipher_ctrl_t *const *p_api_ctrl*, sf_crypto_cipher_cfg_t const *const *p_cfg*)

SSP Crypto Cipher Framework Open operation.

The SF_CRYTO_CIPHER_Open function: Allocates memory required for the cipher operations based on the configuration parameters. Acquires lock for the shared crypto resources. Gets the interface to the HAL driver based on the config parameters. Calls the .open function of the HAL API. On successful open, the module status is updated as such. The shared resources are unlocked before exit.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	One or more input parameters maybe NULL.
SSP_ERR_INVALID_ARGUMENT	An invalid argument is used.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto common control block and the interface.

Get the pointers to TRNG control and API structures from the instance pointed to by the config structure.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Check if the Cipher module has been already opened. If opened, return an error.

Validate the cfg parameters

Fill the control block with the validated cfg parameters

Determine and fill the algorithm type in the control block.

Note: sf_crypto.lock() is not called because the HAL driver does not access HW during open. The HAL interfaceGet API is also called which does not access HW either. If HW will be accessed through the HAL driver, lock should be acquired.

Allocate memory resources used by this framework instance and open HAL driver

◆ SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES()

__STATIC_INLINE uint32_t SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES (uint32_t *key_size_bits*)

Routine to calculate and return the Encryption block size based on the key size.

Private Functions.

Parameters

[in]	key_size_bits	key size in bits.

◆ sf_crypto_cipher_rsa_final()

ssp_err_t sf_crypto_cipher_rsa_final (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Routine to process the final encryption / decryption operation for RSA. This function is called by SF_CRYPTO_CIPHER_CipherFinal().

Return values

SSP_SUCCESS	The cipher update operation was successful.
SSP_ERR_UNSUPPORTED	The module does not support the algorithm bbased on the key type specified by the user.

Returns

See Common Error Codes for other possible return codes.

Clean up the data in the context buffer before exiting.

◆ sf_crypto_cipher_rsa_init()

ssp_err_t sf_crypto_cipher_rsa_init (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_cipher_op_mode_t *cipher_operation_mode*, sf_crypto_key_t const *const *p_key*, sf_crypto_cipher_algorithm_init_params_t * *p_algorithm_specific_params*)

Routine to handle the cipherInit RSA operation.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in]	cipher_operation_mode	The cipher operation mode - encrypt / decrypt.
[in]	p_key	Pointer to the key to be used for the cipher operations.
[in]	p_algorithm_specific_params	Pointer to the algorithm specific parameters.

Return values

SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

All buffers within the context buffer are allocated at Open. Now just zeroise their contents.

Copy data into the control block / context buffer.

◆ sf_crypto_cipher_rsa_interface_get()

void sf_crypto_cipher_rsa_interface_get (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*)

Subroutine to get a RSA HAL API instance. This function is called by sf_crypto_cipher_open_rsa().

Parameters

[in,out]	p_ctrl	Pointer to a Key framework control block, whose p_hal_api filled with HAL RSA interface. This indicates NULL, for not supported MCUs

Get a Crypto common control block and the HAL instance.

Check the RSA key type.

Check the RSA key type.

Get the HAL API instance for a selected algorithm type.

◆ SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES()

__STATIC_INLINE uint32_t SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES (uint32_t *key_size_bits*)

Routine to calculate and return the Encryption block size based on the key size.

Parameters

[in]	key_size_bits	key size in bits.

◆ sf_crypto_cipher_rsa_update()

ssp_err_t sf_crypto_cipher_rsa_update (sf_crypto_cipher_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*, sf_crypto_data_handle_t *const *p_data_out*)

Subroutine to update the cipher RSA operation. This algorithm is only suitable for messages of limited length. The total number of input bytes processed during encryption may not be more than k-11, where k is the RSA key's modulus size in bytes. The encryption block(EB) during encryption with a Public key is built as follows: EB = 00 || 02 || PS || 00 || M :: M (input bytes) is the plaintext message :: PS is an octet string of length k-3-||M|| of pseudo random nonzero octets. The length of PS must be at least 8 octets. :: k is the RSA modulus size.

Parameters

[in,out]	p_ctrl	Pointer to the cipher framework module control block used in the open() call.
[in]	p_data_in	Pointer to the input data structure - has the pointer to input data and the data length
[in,out]	p_data_out	Pointer to the output data structure - has the pointer to output data and the buffer length on input. If data is filled the length is updated on output.

Return values

SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_UNSUPPORTED	Unknown cipher operation mode was passed in.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

input parameters validation - done at the entry nothing to do here no output on update - so no checks for output buffer / length collect input up to the block size for encryption when no padding is selected. collect data up to block size -11 for encryption when padding is selected collect input up to the block size for decryption if it exceeds the above return error

Copy the data into the partial block

◆ SF_CRYPTO_CIPHER_VersionGet()

ssp_err_t SF_CRYPTO_CIPHER_VersionGet (ssp_version_t *const *p_version*)
Gets driver version based on compile time macros. **Return values** SSP_SUCCESS — The version is returned successfully. SSP_ERR_ASSERTION — The parameter p_version is NULL.

sf_crypto_cipher_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Cipher Framework

#include <sf_crypto_cipher.h>

Data Fields

sf_crypto_key_type_t	key_type Key type.
sf_crypto_key_size_t	key_size Key size.
sf_crypto_cipher_mode_t	cipher_chaining_mode Chaining mode specified for the cipher operation.
sf_crypto_cipher_state_t	status Module status.
crypto_algorithm_type_t	cipher_algorithm_type Cipher algorithm for the keys selected.
sf_crypto_instance_ctrl_t *	p_lower_lvl_fwk_common_ctrl Pointer to the Crypto Framework Common instance.

sf_crypto_api_t *	p_lower_lvl_fwk_common_api Pointer to the Crypto Framework Common API.
sf_crypto_trng_instance_ctrl_t *	p_lower_lvl_sf_crypto_trng_ctrl Pointer to the Crypto TRNG API.
sf_crypto_trng_api_t *	p_sf_crypto_trng_api Pointer to the Crypto TRNG API.
void *	p_hal_ctrl Pointer to HAL control structure for the Cipher operation.
void *	p_hal_api Pointer to HAL API structure for the cipher algorithm.
void *	p_cipher_context_buffer Cipher context buffer after DWORD alignment.

Detailed Description

SSP Crypto Cipher Framework instance control block. DO NOT INITIALIZE. Initialization occurs when SF_CRYPTO_CIPHER_Open is called

---

The documentation for this struct was generated from the following file:

- sf_crypto_cipher.h

5.1.3.16 SSP Crypto Hash Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto HASH Framework Module. More...

Data Structures

struct	sf_crypto_hash_instance_ctrl_t

Macros

#define	SF_CRYPTO_HASH_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_CRYPTO_HASH_Open (sf_crypto_hash_ctrl_t *const p_api_ctrl, sf_crypto_hash_cfg_t const *const p_cfg) SSP Crypto HASH Framework Open operation. More...
ssp_err_t	SF_CRYPTO_HASH_Close (sf_crypto_hash_ctrl_t *const p_api_ctrl) SSP Crypto HASH Framework Close operation. More...
ssp_err_t	SF_CRYPTO_HASH_MessageDigestInit (sf_crypto_hash_ctrl_t *const p_api_ctrl) SSP Crypto HASH Framework - Generates the initial message digest in an internal context buffer. Can be called once messageDigestFinal() is called to initialize a new digest operation. Unless a different HASH type is used, users do not need to close the module for new digest operation. This is a blocking call. More...
ssp_err_t	SF_CRYPTO_HASH_MessageDigestUpdate (sf_crypto_hash_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_data_in) SSP Crypto HASH Framework - Hashes input data and saves it in an internal context buffer. Can be called multiple times for additional blocks of data. This is a blocking call. More...
ssp_err_t	SF_CRYPTO_HASH_MessageDigestFinal (sf_crypto_hash_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t *const p_msg_digest, uint32_t *p_size) SSP Crypto HASH Framework - Hashes the last block of data and returns the message digest in the output buffer. Once this function is called, no additional function calls can be made but open function call can be made to initialize a new digest operation. The Output buffer will contain the message digest on success and the buffer length will be updated to reflect the size of the digest. On error, only the length is set to 0.This is a blocking call. More...
ssp_err_t	SF_CRYPTO_HASH_VersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. More...

Detailed Description

RTOS-integrated Crypto HASH Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_HASH_CODE_VERSION_MAJOR

#define SF_CRYPTO_HASH_CODE_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework

Function Documentation

◆ SF_CRYPTO_HASH_Close()

ssp_err_t SF_CRYPTO_HASH_Close (sf_crypto_hash_ctrl_t *const *p_api_ctrl*)

SSP Crypto HASH Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	HASH algorithms are not supported for the MCU part.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto Framework common control block and the interface.

Check if the Crypto HASH Framework has been opened.

Delete memory resources used by this module and close the HASH HAL module.

Mark the module status as 'Closed'.

Decrement the open counter to enable users to close SF_CRYPTO module.

◆ SF_CRYPTO_HASH_MessageDigestFinal()

ssp_err_t SF_CRYPTO_HASH_MessageDigestFinal (sf_crypto_hash_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t *const *p_msg_digest*, uint32_t * *p_size*)

SSP Crypto HASH Framework - Hashes the last block of data and returns the message digest in the output buffer. Once this function is called, no additional function calls can be made but open function call can be made to initialize a new digest operation. The Output buffer will contain the message digest on success and the buffer length will be updated to reflect the size of the digest. On error, only the length is set to 0.This is a blocking call.

Return values

SSP_SUCCESS	The module updated a message Digest successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_INVALID_SIZE	The memory size to store a message digest is not sufficient for the digest type.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	HASH algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_HASH_DIGEST_UPDATED.

Returns

See Common Error Codes for other possible return codes.

Note

p_msg_digest->p_data must be WORD aligned. The memory allocation to store a message digest is user's responsibility.

Get a Crypto Framework common control block and the interface.

Check if the Crypto Framework has been in 'Digest Updated' status. If not, return an error.

Update the message digest.

We are all set now. A message digest is returned. Mark the module status as 'Opened'.

Unlock the module.

◆ SF_CRYPTO_HASH_MessageDigestInit()

ssp_err_t SF_CRYPTO_HASH_MessageDigestInit (sf_crypto_hash_ctrl_t *const *p_api_ctrl*)

SSP Crypto HASH Framework - Generates the initial message digest in an internal context buffer. Can be called once messageDigestFinal() is called to initialize a new digest operation. Unless a different HASH type is used, users do not need to close the module for new digest operation. This is a blocking call.

Return values

SSP_SUCCESS	The module updated a message Digest successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Returns

See Common Error Codes for other possible return codes.

Note

When this function is called if SF_CRYPTO_HASH_DIGEST_INITIALIZED or SF_CRYPTO_HASH_DIGEST_UPDATED, the message digest will be initialized.

Check if the Crypto HASH Framework module has been opened. If not yet opened, return an error.

Create initial message digest code in the module context.

Mark the module status as 'Digest Initialized'.

◆ SF_CRYPTO_HASH_MessageDigestUpdate()

ssp_err_t SF_CRYPTO_HASH_MessageDigestUpdate (sf_crypto_hash_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_data_in*)

SSP Crypto HASH Framework - Hashes input data and saves it in an internal context buffer. Can be called multiple times for additional blocks of data. This is a blocking call.

Return values

SSP_SUCCESS	The module updated a message Digest successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	HASH algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_HASH_DIGEST_INITIALIZED.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto Framework common control block and the interface.

Check if the Crypto Framework has been in 'Digest Initialized' status. If not, return an error.

Update the message digest.

Mark the module status as 'Digest Updated'.

Unlock the module.

◆ SF_CRYPTO_HASH_Open()

ssp_err_t SF_CRYPTO_HASH_Open (sf_crypto_hash_ctrl_t *const *p_api_ctrl*, sf_crypto_hash_cfg_t const *const *p_cfg*)

SSP Crypto HASH Framework Open operation.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	HASH algorithms are not supported for the MCU part.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto Framework common instance, the control block and interface.

Get a lower-level HASH instance.

Check if the Crypto Framework common instance has been opened. If not yet opened, return an error.

Check if the Crypto HASH Framework module has been opened.

Create memory resources used by this module and open a HASH HAL module.

Increment the open counter.

We have successfully processed the open procedures. Mark the module status as 'Opened'.

◆ SF_CRYPTO_HASH_VersionGet()

ssp_err_t SF_CRYPTO_HASH_VersionGet (ssp_version_t *const *p_version*)

Gets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

sf_crypto_hash_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Hash Framework

#include <sf_crypto_hash.h>

Data Fields

sf_crypto_hash_state_t	status Module status.
sf_crypto_hash_type_t	hash_type HASH algorithm type.
sf_crypto_hash_context_t	hash_context Context for calculating message digest.
sf_crypto_instance_t *	p_lower_lvl_crypto_common Pointer to a Crypto Framework common instance.
hash_instance_t *	p_lower_lvl_instance pointer to lower-level crypto module control structure

Detailed Description

SSP Crypto HASH Framework instance control block

The documentation for this struct was generated from the following file:

- sf_crypto_hash.h

5.1.3.17 SSP Crypto Key Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto Key Framework Module. More...

Data Structures

struct	sf_crypto_key_instance_ctrl_t

Macros

#define	SF_CRYPTO_KEY_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	sf_crypto_key_open_aes (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_cfg_t const *const p_cfg) Subroutine to open a Crypto AES HAL module. This function is called by SF_CRYPTO_KEY_Open(). More...
ssp_err_t	sf_crypto_key_close_aes (sf_crypto_key_instance_ctrl_t *p_ctrl) Subroutine to close a Crypto AES HAL module. This function is called by SF_CRYPTO_KEY_Close(). More...
ssp_err_t	sf_crypto_key_generate_aes (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_t *const p_secret_key) Subroutine to generate a AES key. This function is called by SF_CRYPTO_KEY_Generate(). More...
void	sf_crypto_key_aes_interface_get (sf_crypto_key_instance_ctrl_t *const p_ctrl) Subroutine to get a AES HAL API instance. This function is called by sf_crypto_key_open_aes(). Key framework control block's p_hal_api field is filled with HAL AES interface. This will be set to NULL if the API interface is not found on the particular MCU. More...
void	sf_crypto_key_aes_get_available_api_interface (sf_crypto_key_instance_ctrl_t *const p_ctrl, crypto_interface_get_param_t *p_param) Subroutine to get first available AES HAL API instance for ECB/CBC/CTR/GCM chaining modes. This function is called by sf_crypto_key_aes_interface_get(). Key framework control block's p_hal_api field is filled with HAL AES interface. This will be set to NULL if the API interface is not found on the particular MCU. More...
ssp_err_t	sf_crypto_key_aes_verify_wrappedkey_buffersize (sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided buffer_length (indicates size of the secret AES Wrapped key) is sufficient or not, before proceeding with key Generate function. This function is called by AES Key Generate function before passing request to Crypto HAL Key generate API call. More...

ssp_err_t	sf_crypto_key_aes_verify_wrapped_xtskey_buffersize (sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided num_words (indicates size of the secret AES Wrapped key for XTS AES chaining mode) is sufficient or not, before proceeding with key operations. This function is called by sf_crypto_key_aes_verify_wrappedkey_buffersize as an internal function call. More...
ssp_err_t	sf_crypto_key_open_ecc (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_cfg_t const *const p_cfg) Subroutine to open a Crypto ECC HAL module. This function is called by SF_CRYPTO_KEY_Open(). The configuration parameters are validated before opening the HAL module. More...
ssp_err_t	sf_crypto_key_close_ecc (sf_crypto_key_instance_ctrl_t *p_ctrl) Subroutine to close a Crypto ECC HAL module. This function is called by SF_CRYPTO_KEY_Close(). More...
ssp_err_t	sf_crypto_key_generate_ecc (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_public_key) Subroutine to generate a ECC key. This function is called by SF_CRYPTO_KEY_Generate(). More...
ssp_err_t	sf_crypto_key_scalar_multiplication_ecc (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_point_on_curve, sf_crypto_key_t *const p_resultant_vector) Subroutine to perform a ECC scalar multiplication. This function is called by SF_CRYPTO_KEY_ECDHCompute(). More...
ssp_err_t	sf_crypto_key_open_rsa (sf_crypto_key_instance_ctrl_t *p_ctrl, sf_crypto_key_cfg_t const *const p_cfg) Subroutine to open a Crypto RSA HAL module. This function is called by SF_CRYPTO_KEY_Open(). More...
ssp_err_t	sf_crypto_key_close_rsa (sf_crypto_key_instance_ctrl_t *p_ctrl) Subroutine to close a Crypto RSA HAL module. This function is called by SF_CRYPTO_KEY_Close(). More...
ssp_err_t	sf_crypto_key_generate_rsa (sf_crypto_key_instance_ctrl_t *p_ctrl,

	sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_public_key) Subroutine to generate a RSA key. This function is called by SF_CRYPTO_KEY_Generate(). More...
void	sf_crypto_key_rsa_interface_get (sf_crypto_key_instance_ctrl_t *const p_ctrl) Subroutine to get a RSA HAL API instance. This function is called by sf_crypto_key_open_rsa(). More...
ssp_err_t	sf_crypto_key_rsa_verify_privatekey_buffersize (sf_crypto_key_type_t key_type, sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA private key) is sufficient or not, before proceeding with key Generate function. This function is called by RSA Key Generate function before passing request to Crypto HAL Key generate API call. More...
ssp_err_t	sf_crypto_key_rsa_verify_publickey_buffersize (sf_crypto_key_type_t key_type, sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Public key) is sufficient or not, before proceeding with key Generate function. This function is called by RSA Key Generate function before passing request to Crypto HAL Key generate API call. More...
ssp_err_t	sf_crypto_key_rsa_verify_plaintext_buffersize (sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Plain text key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call. More...
ssp_err_t	sf_crypto_key_rsa_verify_crtplaintext_buffersize (sf_crypto_key_size_t key_size, uint32_t buffer_length) Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA CRT Plain text key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call. More...
ssp_err_t	sf_crypto_key_rsa_verify_wrapped_buffersize (sf_crypto_key_size_t key_size, uint32_t buffer_length)

	Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Wrapped key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call. More...
ssp_err_t	SF_CRYPTO_KEY_Open (sf_crypto_key_ctrl_t *const p_api_ctrl, sf_crypto_key_cfg_t const *const p_cfg) SSP Crypto KeyFramework Open operation. More...
ssp_err_t	SF_CRYPTO_KEY_Close (sf_crypto_key_ctrl_t *const p_api_ctrl) SSP Crypto Key Framework Close operation. More...
ssp_err_t	SF_CRYPTO_KEY_Generate (sf_crypto_key_ctrl_t *const p_api_ctrl, sf_crypto_key_t *const p_secret_key, sf_crypto_key_t *const p_public_key) SSP Crypto Framework Key Generate operation. More...
ssp_err_t	SF_CRYPTO_KEY_EcdhSharedSecretCompute (sf_crypto_key_ctrl_t *const p_api_ctrl, sf_crypto_key_t *const p_local_secret_key, sf_crypto_key_t *const p_remote_public_key, sf_crypto_key_t *const p_shared_secret) SSP Crypto Framework ECDH Shared Secret Computation operation. More...
ssp_err_t	SF_CRYPTO_KEY_VersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. More...

Detailed Description

RTOS-integrated Crypto Key Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_KEY_CODE_VERSION_MAJOR

#define SF_CRYPTO_KEY_CODE_VERSION_MAJOR (2U)
The API version of SSP Crypto Framework

Function Documentation

◆ sf_crypto_key_aes_get_available_api_interface()

void sf_crypto_key_aes_get_available_api_interface (sf_crypto_key_instance_ctrl_t *const *p_ctrl*, crypto_interface_get_param_t * *p_param*)

Subroutine to get first available AES HAL API instance for ECB/CBC/CTR/GCM chaining modes. This function is called by sf_crypto_key_aes_interface_get(). Key framework control block's p_hal_api field is filled with HAL AES interface. This will be set to NULL if the API interface is not found on the particular MCU.

Parameters

[in,out]	p_ctrl	Pointer to a Key framework control block.
[in]	p_param	Interface get param. Algorithm type, key type and key size is set by the caller.

◆ sf_crypto_key_aes_interface_get()

void sf_crypto_key_aes_interface_get (sf_crypto_key_instance_ctrl_t *const *p_ctrl*)

Subroutine to get a AES HAL API instance. This function is called by sf_crypto_key_open_aes(). Key framework control block's p_hal_api field is filled with HAL AES interface. This will be set to NULL if the API interface is not found on the particular MCU.

Parameters

[in,out]	p_ctrl	Pointer to a Key framework control block. interface. This indicates NULL, for not supported MCUs

◆ sf_crypto_key_aes_verify_wrapped_xtskey_buffersize()

ssp_err_t sf_crypto_key_aes_verify_wrapped_xtskey_buffersize (sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided num_words (indicates size of the secret AES Wrapped key for XTS AES chaining mode) is sufficient or not, before proceeding with key operations. This function is called by sf_crypto_key_aes_verify_wrappedkey_buffersize as an internal function call.

Parameters

[in]	key_size	Indicates AES key sizes - 128/256-bits, supported chaining mode - XTS
[in]	buffer_length	Length of the secret key which shall be filled by AES Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with AES Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of AES

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_aes_verify_wrappedkey_buffersize()

ssp_err_t sf_crypto_key_aes_verify_wrappedkey_buffersize (sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (indicates size of the secret AES Wrapped key) is sufficient or not, before proceeding with key Generate function. This function is called by AES Key Generate function before passing request to Crypto HAL Key generate API call.

Parameters

[in]	key_size	Indicates AES key sizes - 128/192/256-bits, supported chaining modes - CBC, ECB, GCM, CTR
[in]	buffer_length	Length of the secret key which shall be filled by AES Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with AES Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of AES

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ SF_CRYPTO_KEY_Close()

ssp_err_t SF_CRYPTO_KEY_Close (sf_crypto_key_ctrl_t *const *p_api_ctrl*)

SSP Crypto Key Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto common control block and the interface.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Close Crypto HAL module.

Decrement the open counter to enable users to close SF_CRYPTO module.

◆ sf_crypto_key_close_aes()

ssp_err_t sf_crypto_key_close_aes (sf_crypto_key_instance_ctrl_t * *p_ctrl*)

Subroutine to close a Crypto AES HAL module. This function is called by SF_CRYPTO_KEY_Close().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block

Return values

SSP_SUCCESS	AES HAL module is successfully closed.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_key_close_ecc()

ssp_err_t sf_crypto_key_close_ecc (sf_crypto_key_instance_ctrl_t * *p_ctrl*)

Subroutine to close a Crypto ECC HAL module. This function is called by SF_CRYPTO_KEY_Close().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block

Return values

SSP_SUCCESS	ECC HAL module is successfully closed.

Returns

See Common Error Codes for other possible return codes.

Close the Crypto HAL driver.

Release the control block memory back to byte pool

◆ sf_crypto_key_close_rsa()

ssp_err_t sf_crypto_key_close_rsa (sf_crypto_key_instance_ctrl_t * *p_ctrl*)

Subroutine to close a Crypto RSA HAL module. This function is called by SF_CRYPTO_KEY_Close().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block

Return values

SSP_SUCCESS	RSA HAL module is successfully closed.

Returns

See Common Error Codes for other possible return codes.

◆ SF_CRYPTO_KEY_EcdhSharedSecretCompute()

ssp_err_t SF_CRYPTO_KEY_EcdhSharedSecretCompute (sf_crypto_key_ctrl_t *const *p_api_ctrl*, sf_crypto_key_t *const *p_local_secret_key*, sf_crypto_key_t *const *p_remote_public_key*, sf_crypto_key_t *const *p_shared_secret*)

SSP Crypto Framework ECDH Shared Secret Computation operation.

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	The module does not support scalar multiplication operation.
SSP_ERR_INVALID_SIZE	The length of the buffer supplied for the key to be generated, is insufficient.

Returns

See Common Error Codes for other possible return codes.

Check if ECDH computation is used with ECC, otherwise return an error.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Lock the module to access to Crypto HAL module.

Perform the scalar multiplication.

Unlock the module.

◆ SF_CRYPTO_KEY_Generate()

ssp_err_t SF_CRYPTO_KEY_Generate (sf_crypto_key_ctrl_t *const *p_api_ctrl*, sf_crypto_key_t *const *p_secret_key*, sf_crypto_key_t *const *p_public_key*)

SSP Crypto Framework Key Generate operation.

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
SSP_ERR_INVALID_SIZE	The length of the buffer supplied for the key to be generated, is insufficient.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Lock the module to access to Crypto HAL module.

Generate a key.

Unlock the module.

◆ sf_crypto_key_generate_aes()

ssp_err_t sf_crypto_key_generate_aes (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_t *const *p_secret_key*)

Subroutine to generate a AES key. This function is called by SF_CRYPTO_KEY_Generate().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block.
[out]	p_secret_key	Pointer to a secret key.

Return values

SSP_SUCCESS	The module created the AES Secret key successfully.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key buffer length is not sufficient for the AES Key generation operation.
SSP_ERR_UNSUPPORTED	Procedure is not supported for the supplied parameters.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_key_generate_ecc()

ssp_err_t sf_crypto_key_generate_ecc (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_t *const *p_secret_key*, sf_crypto_key_t *const *p_public_key*)

Subroutine to generate a ECC key. This function is called by SF_CRYPTO_KEY_Generate().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block
[out]	p_secret_key	Pointer to a secret key
[out]	p_public_key	Pointer to a public key

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key buffer length is not sufficient for the ECC Key generation operation.
SSP_ERR_UNSUPPORTED	Procedure is not supported for the supplied parameters.

Returns

See Common Error Codes for other possible return codes.

Verify the domain parameter input buffer size. The length has to be exactly as specified

Verify the public key buffer size that holds the generated public key.

Verify the secret key buffer size that holds the generated private key.

Local variable created to call the HAL API.

Note that the HAL driver requires data in WORDS

Call Crypto HAL driver.

◆ sf_crypto_key_generate_rsa()

ssp_err_t sf_crypto_key_generate_rsa (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_t *const *p_secret_key*, sf_crypto_key_t *const *p_public_key*)

Subroutine to generate a RSA key. This function is called by SF_CRYPTO_KEY_Generate().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block
[out]	p_secret_key	Pointer to a secret key
[out]	p_public_key	Pointer to a public key

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key buffer length is not sufficient for the RSA Key generation operation.
SSP_ERR_UNSUPPORTED	Procedure is not supported for the supplied parameters.

Returns

See Common Error Codes for other possible return codes.

◆ SF_CRYPTO_KEY_Open()

ssp_err_t SF_CRYPTO_KEY_Open (sf_crypto_key_ctrl_t *const *p_api_ctrl*, sf_crypto_key_cfg_t const *const *p_cfg*)

SSP Crypto KeyFramework Open operation.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
SSP_ERR_INVALID_SIZE	The buffer length of one of the configuration parameters is invalid.

Returns

See Common Error Codes for other possible return codes.

Get a Crypto common control block and the interface.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Check if the Crypto key framework module has been already opened. If yes, return an error.

Set a key type, size, domain and generator point. Generator point and domain are UNUSED for RSA and AES algorithms, applicable only for ECC.

Open Crypto HAL module.

◆ sf_crypto_key_open_aes()

ssp_err_t sf_crypto_key_open_aes (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_cfg_t const *const *p_cfg*)

Subroutine to open a Crypto AES HAL module. This function is called by SF_CRYPTO_KEY_Open().

Parameters

[in,out]	p_ctrl	Pointer to a Crypto Key Framework module control block
[in]	p_cfg	Pointer to a Crypto Key Framework module configuration structure

Return values

SSP_SUCCESS	AES HAL module is successfully opened.
SSP_ERR_OUT_OF_MEMORY	Failed to allocate memory to store AES HAL module control block.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_key_open_ecc()

ssp_err_t sf_crypto_key_open_ecc (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_cfg_t const *const *p_cfg*)

Subroutine to open a Crypto ECC HAL module. This function is called by SF_CRYPTO_KEY_Open(). The configuration parameters are validated before opening the HAL module.

Parameters

[in,out]	p_ctrl	Pointer to a Crypto Key Framework module control block
[in,out]	p_cfg	Pointer to a Crypto Key Framework module configuration structure

Return values

SSP_SUCCESS	ECC HAL module is successfully opened.
SSP_ERR_OUT_OF_MEMORY	Failed to allocate memory to store ECC HAL module control block.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes for other possible return codes.

Validate the cfg parameters. The key type is already validated.

Verify the key size.

Verify the domain parameter input buffer size. The length has to be exactly as specified

Verify the generator point input buffer size. The length has to be exactly as specified

Get a Crypto common control block and the interface.

Allocate memory for a Crypto HAL control block in the byte pool.

Get a ECC interface instance.

Set Crypto HAL API instance with the control common hardware api.

Open the Crypto HAL module.

◆ sf_crypto_key_open_rsa()

ssp_err_t sf_crypto_key_open_rsa (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_cfg_t const *const *p_cfg*)

Subroutine to open a Crypto RSA HAL module. This function is called by SF_CRYPTO_KEY_Open().

Parameters

[in,out]	p_ctrl	Pointer to a Crypto Key Framework module control block
[in,out]	p_cfg	Pointer to a Crypto Key Framework module configuration structure

Return values

SSP_SUCCESS	RSA HAL module is successfully opened.
SSP_ERR_OUT_OF_MEMORY	Failed to allocate memory to store RSA HAL module control block.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_key_rsa_interface_get()

void sf_crypto_key_rsa_interface_get (sf_crypto_key_instance_ctrl_t *const *p_ctrl*)

Subroutine to get a RSA HAL API instance. This function is called by sf_crypto_key_open_rsa().

Parameters

[in,out]	p_ctrl	Pointer to a Key framework control block, whose p_hal_api filled with HAL RSA interface. This indicates NULL, for not supported MCUs

◆ sf_crypto_key_rsa_verify_crtplaintext_buffersize()

ssp_err_t sf_crypto_key_rsa_verify_crtplaintext_buffersize (sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA CRT Plain text key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call.

Parameters

[in]	key_size	Indicates RSA key size - 1024/2048-bits.
[in]	buffer_length	Length of the secret key which shall be filled by RSA Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with RSA Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of RSA.

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_rsa_verify_plaintext_buffersize()

ssp_err_t sf_crypto_key_rsa_verify_plaintext_buffersize (sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Plain text key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call.

Parameters

[in]	key_size	Indicates RSA key size - 1024/2048-bits.
[in]	buffer_length	Length of the secret key which shall be filled by RSA Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with RSA Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of RSA.

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_rsa_verify_privatekey_buffersize()

ssp_err_t sf_crypto_key_rsa_verify_privatekey_buffersize (sf_crypto_key_type_t *key_type*, sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA private key) is sufficient or not, before proceeding with key Generate function. This function is called by RSA Key Generate function before passing request to Crypto HAL Key generate API call.

Parameters

[in]	key_type	Indicates RSA key type - Plain text/CRT Plain Text/ Wrapped key.
[in]	key_size	Indicates RSA key size - 1024/2048-bits.
[in]	buffer_length	Length of the secret key which shall be filled by RSA Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with RSA Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of RSA.

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_rsa_verify_publickey_buffersize()

ssp_err_t sf_crypto_key_rsa_verify_publickey_buffersize (sf_crypto_key_type_t *key_type*, sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Public key) is sufficient or not, before proceeding with key Generate function. This function is called by RSA Key Generate function before passing request to Crypto HAL Key generate API call.

Parameters

[in]	key_type	Indicates RSA key type - Plain text/CRT Plain Text/ Wrapped key.
[in]	key_size	Indicates RSA key size - 1024/2048-bits.
[in]	buffer_length	Length of the secret key which shall be filled by RSA Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with RSA Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of RSA.

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_rsa_verify_wrapped_buffersize()

ssp_err_t sf_crypto_key_rsa_verify_wrapped_buffersize (sf_crypto_key_size_t *key_size*, uint32_t *buffer_length*)

Subroutine to verify that the provided buffer_length (buffer to hold the secret RSA Wrapped key) is sufficient or not, before proceeding with key Generate function. This function is called by sf_crypto_key_rsa_verify_privatekey_buffersize as an internal function call.

Parameters

[in]	key_size	Indicates RSA key size - 1024/2048-bits.
[in]	buffer_length	Length of the secret key which shall be filled by RSA Key generation algorithm.

Return values

SSP_SUCCESS	Key length is successful, and proceed with RSA Key Generation.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key length is not sufficient for the Key generation operation of RSA.

Note

This function is not a user API but an internal function for SF_CRYPTO_KEY to verify whether passed buffer size is sufficient for Key to hold in Key Generate API call.

◆ sf_crypto_key_scalar_multiplication_ecc()

ssp_err_t sf_crypto_key_scalar_multiplication_ecc (sf_crypto_key_instance_ctrl_t * *p_ctrl*, sf_crypto_key_t *const *p_secret_key*, sf_crypto_key_t *const *p_point_on_curve*, sf_crypto_key_t *const *p_resultant_vector*)

Subroutine to perform a ECC scalar multiplication. This function is called by SF_CRYPTO_KEY_ECDHCompute().

Parameters

[in]	p_ctrl	Pointer to a Crypto Key Framework module control block
[in]	p_secret_key	Pointer to a secret key
[in]	p_point_on_curve	Pointer to a point specified on the curve.
[in]	p_resultant_vector	Pointer to the resultant point on the curve.

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_INVALID_SIZE	Failed, as the allocated key buffer length is not sufficient for the ECC Key generation operation.
SSP_ERR_UNSUPPORTED	Procedure is not supported for the supplied parameters.

Returns

See Common Error Codes for other possible return codes.

Verify the secret key buffer size that holds the generated private key.

Verify the input buffer size for holding the data, representing point on the curve data.

Verify the output buffer size to hold the resultant data, representing point on the curve.

Local variable created to call the HAL API.

Note that the HAL driver requires data in WORDS

Call Crypto HAL driver.

◆ SF_CRYPTO_KEY_VersionGet()

ssp_err_t SF_CRYPTO_KEY_VersionGet (ssp_version_t *const *p_version*)
Gets driver version based on compile time macros. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — The parameter p_version is NULL.

sf_crypto_key_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Key Framework

#include <sf_crypto_key.h>

Data Fields

sf_crypto_key_state_t	status Module status.
sf_crypto_key_type_t	key_type Key type.
sf_crypto_key_size_t	key_size Key size.
sf_crypto_data_handle_t	domain_params Domain parameters structure with pointer to data and length.
sf_crypto_data_handle_t	generator_point Generator Point structure with pointer to data and length.
sf_crypto_instance_ctrl_t *	p_fwk_common_ctrl Pointer to the Crypto Framework Common instance.

sf_crypto_api_t *	p_fwk_common_api Pointer to the Crypto Framework Common instance.
void *	p_hal_ctrl pointer to Crypto module control structure
void *	p_hal_api pointer to Crypto module API structure

Detailed Description

SSP Crypto Key Framework instance control block

The documentation for this struct was generated from the following file:

- sf_crypto_key.h

5.1.3.18 SSP Crypto Key Installation Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto Key Installation Framework Module. More...

Data Structures

struct	sf_crypto_key_installation_instance_ctrl_t

Macros

#define	SF_CRYPTO_KEY_INSTALLATION_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_CRYPTO_KEY_INSTALLATION_Open (sf_crypto_key_installation_ctrl_t *const p_api_ctrl, sf_crypto_key_installation_cfg_t const *const p_cfg) SSP Crypto Key Installation Framework Open operation. More...
ssp_err_t	SF_CRYPTO_KEY_INSTALLATION_Close (sf_crypto_key_installation_ctrl_t *const p_api_ctrl)

	SSP Crypto Key Installation Framework Close operation. More...
ssp_err_t	SF_CRYPTO_KEY_INSTALLATION_KeyInstall (sf_crypto_key_installation_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_user_key_rsa_modulus, sf_crypto_data_handle_t const *const p_user_key_input, sf_crypto_key_installation_shared_index_t const shared_index_input, sf_crypto_data_handle_t const *const p_session_key_input, uint32_t const *const p_iv_input, sf_crypto_data_handle_t *const p_key_data_out) SSP Crypto Framework Key Installation operation. More...
ssp_err_t	SF_CRYPTO_KEY_INSTALLATION_VersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. More...

Detailed Description

RTOS-integrated Crypto Key Installation Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_KEY_INSTALLATION_CODE_VERSION_MAJOR

```
#define SF_CRYPTO_KEY_INSTALLATION_CODE_VERSION_MAJOR   (2U)
```

The API version of SSP Crypto Key Installation Framework

Function Documentation

◆ SF_CRYPTO_KEY_INSTALLATION_Close()

ssp_err_t SF_CRYPTO_KEY_INSTALLATION_Close (sf_crypto_key_installation_ctrl_t *const *p_api_ctrl*)

SSP Crypto Key Installation Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Call HAL API to Close Key Installation HAL Driver

◆ SF_CRYPTO_KEY_INSTALLATION_KeyInstall()

ssp_err_t SF_CRYPTO_KEY_INSTALLATION_KeyInstall (sf_crypto_key_installation_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_user_key_rsa_modulus*, sf_crypto_data_handle_t const *const *p_user_key_input*, sf_crypto_key_installation_shared_index_t const *shared_index_input*, sf_crypto_data_handle_t const *const *p_session_key_input*, uint32_t const *const *p_iv_input*, sf_crypto_data_handle_t *const *p_key_data_out*)

SSP Crypto Framework Key Installation operation.

Return values

SSP_SUCCESS	The module created a key successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_INVALID_ARGUMENT	At least one of the input arguments is invalid.
SSP_ERR_INVALID_SIZE	At least one of the buffers provided is of invalid size.

Returns

See Common Error Codes for other possible return codes.

Validate modulus and output buffers only in case of RSA key type.

Check if the Crypto Key Installation Framework has been opened. If not yet opened, return an error.

Call HAL API to perform Key Installation operation

◆ SF_CRYPTO_KEY_INSTALLATION_Open()

ssp_err_t SF_CRYPTO_KEY_INSTALLATION_Open (sf_crypto_key_installation_ctrl_t *const *p_api_ctrl*, sf_crypto_key_installation_cfg_t const *const *p_cfg*)

SSP Crypto Key Installation Framework Open operation.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_INVALID_ARGUMENT	The module does not support the key type specified by user.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Check if SF Crypto Key Installation is already opened

Validate and copy configuration parameters to control block

Call HAL API to Open Key Installation HAL Driver

Set Key Installation Framework module status to Open

◆ SF_CRYPTO_KEY_INSTALLATION_VersionGet()

ssp_err_t SF_CRYPTO_KEY_INSTALLATION_VersionGet (ssp_version_t *const *p_version*)

Gets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

sf_crypto_key_installation_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Key Installation Framework

#include <sf_crypto_key_installation.h>

Data Fields

sf_crypto_key_installation_state_t	status Module status.
sf_crypto_key_type_t	key_type Type of key to be installed.
sf_crypto_key_size_t	key_size Size of key to be installed.
sf_crypto_instance_ctrl_t *	p_lower_lvl_common_ctrl Pointer to the Crypto Framework Common instance.
sf_crypto_api_t *	p_lower_lvl_common_api Pointer to the Crypto Framework Common instance.
void *	p_lower_lvl_instance Pointer to HAL KeyInstall Crypto module instance structure.

Detailed Description

SSP Crypto Key Installation Framework instance control block

The documentation for this struct was generated from the following file:

- sf_crypto_key_installation.h

5.1.3.19 SSP Crypto Signature Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto Signature Framework Module. More...

Data Structures

struct	sf_crypto_signature_context_t
struct	sf_crypto_signature_instance_ctrl_t

Macros

#define	SF_CRYPTO_SIGNATURE_CODE_VERSION_MAJOR (2U)

Enumerations

enum	sf_crypto_signature_state_t { SF_CRYPTO_SIGNATURE_CLOSED, SF_CRYPTO_SIGNATURE_OPENED }
enum	sf_crypto_signature_operation_state_t { SF_CRYPTO_SIGNATURE_OPERATION_STATE_OPEN, SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_INITIALIZED, SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_UPDATED, SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_FINALIZED, SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_INITIALIZED, SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_UPDATED, SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_FINALIZED }

Functions

ssp_err_t	sf_crypto_signature_key_size_config_rsa (sf_crypto_signature_cfg_t const *const p_cfg) Function for Crypto Signature Framework to check configuration params - Key size. This function is called by sf_crypto_signature_validate_config(). More...
ssp_err_t	sf_crypto_signature_open_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_signature_cfg_t const *const p_cfg) Function for Crypto Signature Framework to open the RSA HAL driver. This function is called by sf_crypto_signature_hal_open(). More...
ssp_err_t	sf_crypto_signature_close_rsa (sf_crypto_signature_instance_ctrl_t *p_ctrl) Function for Crypto Signature Framework to close the RSA HAL driver. This function is called by sf_crypto_signature_hal_close(). More...
ssp_err_t	sf_crypto_signature_context_init_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_signature_mode_t operation_mode, sf_crypto_signature_algorithm_init_params_t *const

	p_algorithm_specific_params, sf_crypto_key_t const *const p_key) Function for Crypto Signature Framework to initialize the Signature framework module context. This function is called by sf_crypto_signature_hal_context_init(). More...
ssp_err_t	sf_crypto_signature_sign_update_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message) Function for Crypto Signature Framework to perform sign update operation. This function is called by sf_crypto_signature_hal_sign_update(). More...
ssp_err_t	sf_crypto_signature_verify_update_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message) Function for Crypto Signature Framework to perform verify update operation. This function is called by sf_crypto_signature_hal_verify_update(). More...
ssp_err_t	sf_crypto_signature_sign_final_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_message, sf_crypto_data_handle_t *const p_dest) Function for Crypto Signature Framework to perform sign final operation. This function is called by sf_crypto_signature_hal_sign_final(). More...
ssp_err_t	sf_crypto_signature_verify_final_rsa (sf_crypto_signature_instance_ctrl_t *const p_ctrl, sf_crypto_data_handle_t const *const p_signature, sf_crypto_data_handle_t const *const p_message) Function for Crypto Signature Framework to perform verify final operation. This function is called by sf_crypto_signature_hal_verify_final(). More...
ssp_err_t	SF_CRYPTO_SIGNATURE_Open (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_signature_cfg_t const *const p_cfg) SSP Crypto Signature Framework Open operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_Close (sf_crypto_signature_ctrl_t *const p_api_ctrl) SSP Crypto Signature Framework Close operation. More...

ssp_err_t	SF_CRYPTO_SIGNATURE_ContextInit (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_signature_mode_t operation_mode, sf_crypto_signature_algorithm_init_params_t *const p_algorithm_specific_params, sf_crypto_key_t const *const p_key) SSP Crypto Signature Framework Context Initialization operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_SignUpdate (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_message) SSP Crypto Signature Framework Signature Update operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_VerifyUpdate (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_message) SSP Crypto Signature Framework Signature-Verification Update operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_SignFinal (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_message, sf_crypto_data_handle_t *const p_dest) SSP Crypto Signature Framework Signature Final operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_VerifyFinal (sf_crypto_signature_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t const *const p_signature, sf_crypto_data_handle_t const *const p_message) SSP Crypto Signature Framework Signature-Verification Update operation. More...
ssp_err_t	SF_CRYPTO_SIGNATURE_VersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. More...
ssp_err_t	sf_crypto_signature_validate_sign_operation_state_transition (sf_crypto_signature_instance_ctrl_t *p_ctrl, sf_crypto_signature_operation_state_t next_state) SSP Crypto Signature Framework State transition validation for sign operation. More...
ssp_err_t	sf_crypto_signature_validate_verify_operation_state_transition (sf_crypto_signature_instance_ctrl_t *p_ctrl, sf_crypto_signature_operation_state_t next_state)

	SSP Crypto Signature Framework State transition validation for verify operation. More...

Detailed Description

RTOS-integrated Crypto Signature Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_SIGNATURE_CODE_VERSION_MAJOR

#define SF_CRYPTO_SIGNATURE_CODE_VERSION_MAJOR (2U)
The API version of SSP Crypto Signature Framework

Enumeration Type Documentation

◆ sf_crypto_signature_operation_state_t

enum sf_crypto_signature_operation_state_t	
Internal state codes for the SSP Crypto Signature framework module.	
Enumerator	
SF_CRYPTO_SIGNATURE_OPERATION_STATE_OPEN	Module opened to perform sign/verify operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_INITIALIZED	Context is initialized for Sign Operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_UPDATED	Sign operation is in progress.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_FINALIZED	Sign operation has been completed.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_INITIALIZED	Context is initialized for Verify Operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_UPDATED	Verify operation is in progress.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_FINALIZED	Verify operation has been completed.

◆ sf_crypto_signature_state_t

enum sf_crypto_signature_state_t

State codes for the SSP Crypto Signature framework module. Once the module is opened successfully, then the state is transition to OPENED state. After sign/verify operations, the Signature framework module must be closed with CLOSED state.

Enumerator	
SF_CRYPTO_SIGNATURE_CLOSED	The Signature module is closed.
SF_CRYPTO_SIGNATURE_OPENED	The Signature module is opened.

Function Documentation

◆ SF_CRYPTO_SIGNATURE_Close()

ssp_err_t SF_CRYPTO_SIGNATURE_Close (sf_crypto_signature_ctrl_t *const *p_api_ctrl*)

SSP Crypto Signature Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Close Crypto HAL module.

Decrement Open counter to indicate a (this) module is closed.

◆ sf_crypto_signature_close_rsa()

ssp_err_t sf_crypto_signature_close_rsa (sf_crypto_signature_instance_ctrl_t * *p_ctrl*)

Function for Crypto Signature Framework to close the RSA HAL driver. This function is called by sf_crypto_signature_hal_close().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_INTERNAL	Critical internal error.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_signature_context_init_rsa()

ssp_err_t sf_crypto_signature_context_init_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_signature_mode_t *operation_mode*, sf_crypto_signature_algorithm_init_params_t *const *p_algorithm_specific_params*, sf_crypto_key_t const *const *p_key*)

Function for Crypto Signature Framework to initialize the Signature framework module context. This function is called by sf_crypto_signature_hal_context_init().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	operation_mode	Perform Sign Or Verify operation.
[in]	p_algorithm_specific_params	Algorithm specific params.
[in]	p_key	Private key if sign operation is to be performed. Public key if verify operation is to be performed.

Return values

SSP_SUCCESS	The module context was successfully initialized.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

◆ SF_CRYPTO_SIGNATURE_ContextInit()

ssp_err_t SF_CRYPTO_SIGNATURE_ContextInit (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_signature_mode_t *operation_mode*, sf_crypto_signature_algorithm_init_params_t *const *p_algorithm_specific_params*, sf_crypto_key_t const *const *p_key*)

SSP Crypto Signature Framework Context Initialization operation.

Return values

SSP_SUCCESS	The module context was successfully initialized.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Setup Context for HAL.

◆ sf_crypto_signature_key_size_config_rsa()

ssp_err_t sf_crypto_signature_key_size_config_rsa (sf_crypto_signature_cfg_t const *const *p_cfg*)

Function for Crypto Signature Framework to check configuration params - Key size. This function is called by sf_crypto_signature_validate_config().

Parameters

[in]	p_cfg	Pointer to sf_crypto_signature_cfg_t configuration structure.

Return values

SSP_SUCCESS	Valid RSA Key size.
SSP_ERR_INVALID_ARGUMENT	Invalid RSA Key size.

◆ SF_CRYPTO_SIGNATURE_Open()

ssp_err_t SF_CRYPTO_SIGNATURE_Open (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_signature_cfg_t const *const *p_cfg*)

SSP Crypto Signature Framework Open operation.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.

Returns

See Common Error Codes for other possible return codes.

Setup control block with common framework module (instance control and API).

Setup control block with Hash framework module (instance).

Check if Crypto Common module is opened before calling this API.

Check if SF Crypto Signature is already opened.

Validate configuration parameters.

Copy configuration parameters to control block.

Open Crypto HAL module.

Set Signature Framework module internal operation state to Open.

Set Signature Framework module status to Open.

Increment Open counter to indicate a (this) module is open.

◆ sf_crypto_signature_open_rsa()

ssp_err_t sf_crypto_signature_open_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_signature_cfg_t const *const *p_cfg*)

Function for Crypto Signature Framework to open the RSA HAL driver. This function is called by sf_crypto_signature_hal_open().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	p_cfg	Pointer to sf_crypto_signature_cfg_t configuration structure.

Return values

SSP_SUCCESS	The module was successfully opened.
SSP_ERR_INTERNAL	Critical internal error.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_signature_sign_final_rsa()

ssp_err_t sf_crypto_signature_sign_final_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_message*, sf_crypto_data_handle_t *const *p_dest*)

Function for Crypto Signature Framework to perform sign final operation. This function is called by sf_crypto_signature_hal_sign_final().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	p_message	Pointer to data handle containing last block of data and its length. If there is no more data to be passed this param can be set to NULL.
[in,out]	p_dest	Pointer to data handle containing pointer to a buffer for storing signature. The data_length of this handle must be populated with the buffer length. Upon successful return this data_length will be updated with the number of bytes written to this buffer.

Return values

SSP_SUCCESS	Sign Final operation was performed successfully.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_INVALID_SIZE	Not enough space to store signature.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_signature_sign_update_rsa()

ssp_err_t sf_crypto_signature_sign_update_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_message*)

Function for Crypto Signature Framework to perform sign update operation. This function is called by sf_crypto_signature_hal_sign_update().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	p_message	Pointer to data handle containing update data and its length.

Return values

SSP_SUCCESS	Sign update was performed successfully.
SSP_ERR_ASSERTION	Critical internal error.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

◆ SF_CRYPTO_SIGNATURE_SignFinal()

ssp_err_t SF_CRYPTO_SIGNATURE_SignFinal (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_message*, sf_crypto_data_handle_t *const *p_dest*)

SSP Crypto Signature Framework Signature Final operation.

Return values

SSP_SUCCESS	Sign Final operation was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_CRYPTO_INVALID_SIZE	Not enough space to store signature.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Validate state transition.

Lock the module to access to Crypto HAL module.

Perform Sign Final.

Set Signature Framework internal state.

Unlock the module.

◆ SF_CRYPTO_SIGNATURE_SignUpdate()

ssp_err_t SF_CRYPTO_SIGNATURE_SignUpdate (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_message*)

SSP Crypto Signature Framework Signature Update operation.

Return values

SSP_SUCCESS	Sign update was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Validate state transition.

Lock the module to access to Crypto HAL module.

Perform Sign Update.

Set Signature Framework internal state.

Unlock the module.

◆ sf_crypto_signature_validate_sign_operation_state_transition()

ssp_err_t sf_crypto_signature_validate_sign_operation_state_transition (sf_crypto_signature_instance_ctrl_t * *p_ctrl*, sf_crypto_signature_operation_state_t *next_state*)

SSP Crypto Signature Framework State transition validation for sign operation.

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	next_state	Requested next state.

Return values

SSP_SUCCESS	Valid call to the calling API.
SSP_ERR_INVALID_CALL	Invalid call to the calling API.

◆ sf_crypto_signature_validate_verify_operation_state_transition()

ssp_err_t sf_crypto_signature_validate_verify_operation_state_transition (sf_crypto_signature_instance_ctrl_t * *p_ctrl*, sf_crypto_signature_operation_state_t *next_state*)

SSP Crypto Signature Framework State transition validation for verify operation.

Parameters

[in]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	next_state	Requested next state.

Return values

SSP_SUCCESS	Valid call to the calling API.
SSP_ERR_INVALID_CALL	Invalid call to the calling API.

◆ sf_crypto_signature_verify_final_rsa()

ssp_err_t sf_crypto_signature_verify_final_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_signature*, sf_crypto_data_handle_t const *const *p_message*)

Function for Crypto Signature Framework to perform verify final operation. This function is called by sf_crypto_signature_hal_verify_final().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	p_signature	Pointer to data handle containing signature and its length.
[in]	p_message	Pointer to data handle containing message in appropriate format and its length.

Return values

SSP_SUCCESS	Verify Final operation was performed successfully.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_INVALID_SIZE	Invalid signature length.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.

Returns

See Common Error Codes for other possible return codes.

◆ sf_crypto_signature_verify_update_rsa()

ssp_err_t sf_crypto_signature_verify_update_rsa (sf_crypto_signature_instance_ctrl_t *const *p_ctrl*, sf_crypto_data_handle_t const *const *p_message*)

Function for Crypto Signature Framework to perform verify update operation. This function is called by sf_crypto_signature_hal_verify_update().

Parameters

[in,out]	p_ctrl	Pointer to Crypto Signature Framework instance control block structure.
[in]	p_message	Pointer to data handle containing message in appropriate format and its length.

Return values

SSP_SUCCESS	Verify update operation was performed successfully.
SSP_ERR_ASSERTION	Critical internal error.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

◆ SF_CRYPTO_SIGNATURE_VerifyFinal()

ssp_err_t SF_CRYPTO_SIGNATURE_VerifyFinal (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_signature*, sf_crypto_data_handle_t const *const *p_message*)

SSP Crypto Signature Framework Signature-Verification Update operation.

Return values

SSP_SUCCESS	Verify final operation is performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_CRYPTO_INVALID_SIZE	Invalid signature length.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Validate state transition.

Lock the module to access to Crypto HAL module.

Perform Verify Final.

Set Signature Framework internal state.

Unlock the module.

◆ SF_CRYPTO_SIGNATURE_VerifyUpdate()

ssp_err_t SF_CRYPTO_SIGNATURE_VerifyUpdate (sf_crypto_signature_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t const *const *p_message*)

SSP Crypto Signature Framework Signature-Verification Update operation.

Return values

SSP_SUCCESS	Verify update operation was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.

Returns

See Common Error Codes for other possible return codes.

Check if the Crypto Framework has been opened. If not yet opened, return an error.

Validate state transition.

Lock the module to access to Crypto HAL module.

Perform Verify Update.

Set Signature Framework internal state.

Unlock the module.

◆ SF_CRYPTO_SIGNATURE_VersionGet()

ssp_err_t SF_CRYPTO_SIGNATURE_VersionGet (ssp_version_t *const *p_version*)

Gets driver version based on compile time macros.

Return values

SSP_SUCCESS	Function returned successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

sf_crypto_signature_context_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Signature Framework

#include <sf_crypto_signature.h>

Data Fields

sf_crypto_signature_mode_t	operation_mode
sf_crypto_signature_algorithm_init_params_t *	p_aglorithm_specific_params
sf_crypto_data_handle_t	buffer
uint8_t *	p_key_data
uint32_t	key_data_length

Detailed Description

Internal SSP Crypto Signature framework module context.

Field Documentation

◆ buffer

sf_crypto_data_handle_t sf_crypto_signature_context_t::buffer
Internal buffer to format input data

◆ key_data_length

uint32_t sf_crypto_signature_context_t::key_data_length
Length of key data.

◆ operation_mode

sf_crypto_signature_mode_t sf_crypto_signature_context_t::operation_mode
Operating mode. (Sign / Verify operation)

◆ p_aglorithm_specific_params

sf_crypto_signature_algorithm_init_params_t* sf_crypto_signature_context_t::p_aglorithm_specific_params
Algorithm specific parameters. OR hold formatted input data.

◆ p_key_data

uint8_t* sf_crypto_signature_context_t::p_key_data
Buffer to hold private key in case of Sign Operations. OR. Buffer to hold public key in case of Verify Operations.

The documentation for this struct was generated from the following file:

- sf_crypto_signature.h

sf_crypto_signature_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SSP Crypto Signature Framework

#include <sf_crypto_signature.h>

Data Fields

sf_crypto_signature_state_t	status Module status.
sf_crypto_key_type_t	key_type Key type.
sf_crypto_key_size_t	key_size Key size.
sf_crypto_signature_operation_state_t	operation_state Internal Operation state.

sf_crypto_signature_context_t	operation_context
sf_crypto_instance_ctrl_t *	p_lower_lvl_common_ctrl
sf_crypto_api_t *	p_lower_lvl_common_api
void *	p_hal_ctrl
void *	p_hal_api
sf_crypto_hash_instance_t *	p_lower_lvl_sf_crypto_hash

Detailed Description

SSP Crypto Signature Framework instance control block

Field Documentation

◆ operation_context

sf_crypto_signature_context_t sf_crypto_signature_instance_ctrl_t::operation_context
Context for sign / verify operations.

◆ p_hal_api

void* sf_crypto_signature_instance_ctrl_t::p_hal_api
pointer to Crypto module API structure

◆ p_hal_ctrl

void* sf_crypto_signature_instance_ctrl_t::p_hal_ctrl
pointer to Crypto module control structure

◆ p_lower_lvl_common_api

sf_crypto_api_t* sf_crypto_signature_instance_ctrl_t::p_lower_lvl_common_api
Pointer to the Crypto Framework API instance

◆ p_lower_lvl_common_ctrl

sf_crypto_instance_ctrl_t* sf_crypto_signature_instance_ctrl_t::p_lower_lvl_common_ctrl
Pointer to the Crypto Framework Common instance

◆ p_lower_lvl_sf_crypto_hash

sf_crypto_hash_instance_t* sf_crypto_signature_instance_ctrl_t::p_lower_lvl_sf_crypto_hash
pointer to Crypto Framework Hash instance

The documentation for this struct was generated from the following file:

- sf_crypto_signature.h

5.1.3.20 SSP Crypto TRNG Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Crypto TRNG Framework Module. More...

Macros

#define	SF_CRYPTO_TRNG_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_CRYPTO_TRNG_Open (sf_crypto_trng_ctrl_t *const p_api_ctrl, sf_crypto_trng_cfg_t const *const p_cfg) SSP Crypto TRNG Framework Open operation. More...
ssp_err_t	SF_CRYPTO_TRNG_Close (sf_crypto_trng_ctrl_t *const p_api_ctrl) SSP Crypto TRNG Framework Close operation. More...
ssp_err_t	SF_CRYPTO_TRNG_RandomNumberGenerate (sf_crypto_trng_ctrl_t *const p_api_ctrl, sf_crypto_data_handle_t *const p_random_number_buff) SSP Crypto TRNG Framework True Random Generation operation. More...

ssp_err_t	SF_CRYPTO_TRNG_VersionGet (ssp_version_t *const p_version) Sets TRNG Framework Code and API version based on compile time macros. More...

Variables

const sf_crypto_trng_api_t	g_sf_crypto_trng_api

Detailed Description

RTOS-integrated Crypto TRNG Framework Module.

Macro Definition Documentation

◆ SF_CRYPTO_TRNG_CODE_VERSION_MAJOR

```
#define SF_CRYPTO_TRNG_CODE_VERSION_MAJOR  (2U)
```

The API version of SSP Crypto Framework

Function Documentation

◆ SF_CRYPTO_TRNG_Close()

ssp_err_t SF_CRYPTO_TRNG_Close (sf_crypto_trng_ctrl_t *const *p_api_ctrl*)

SSP Crypto TRNG Framework Close operation.

Return values

SSP_SUCCESS	The module was successfully closed
SSP_ERR_ASSERTION	One or more input parameters are NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Common module has not been opened.
SSP_ERR_NOT_OPEN	The module has not been opened before calling this API.

Returns

See Common Error Codes for other possible return codes.

Check if SF Crypto TRNG is opened before

Call HAL API to Close TRNG HAL Driver

Set TRNG status to Closed

Decrement Open counter to indicate a (this) module is closed

◆ SF_CRYPTO_TRNG_Open()

ssp_err_t SF_CRYPTO_TRNG_Open (sf_crypto_trng_ctrl_t *const *p_api_ctrl*, sf_crypto_trng_cfg_t const *const *p_cfg*)

SSP Crypto TRNG Framework Open operation.

Return values

SSP_SUCCESS	The module was successfully opened
SSP_ERR_ASSERTION	One or more input parameters are NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto framework common module has not been opened before calling this API.
SSP_ERR_ALREADY_OPEN	The module has already been opened.

Returns

See Common Error Codes for other possible return codes.

Check if Crypto Common module is opened before calling this API

Check if SF Crypto TRNG is already opened

Setup HAL API

Call HAL API to Open TRNG HAL Driver

Set TRNG status to Open

Increment Open counter to indicate a (this) module is open

◆ SF_CRYPTO_TRNG_RandomNumberGenerate()

ssp_err_t SF_CRYPTO_TRNG_RandomNumberGenerate (sf_crypto_trng_ctrl_t *const *p_api_ctrl*, sf_crypto_data_handle_t *const *p_random_number_buff*)

SSP Crypto TRNG Framework True Random Generation operation.

Return values

SSP_SUCCESS	The module was successfully closed.
SSP_ERR_NOT_OPEN	The module has not been opened before calling this API.
SSP_ERR_ASSERTION	One or more input parameters are NULL.

Returns

See Common Error Codes for other possible return codes.

Check if SF Crypto TRNG is opened before

Acquire SF Crypto Common lock before accessing HAL

Call HAL API to generate true random number

Get 16-byte multiples of TRNs first and store/hold it in the user buffer

Release SF Crypto Common lock

◆ SF_CRYPTO_TRNG_VersionGet()

ssp_err_t SF_CRYPTO_TRNG_VersionGet (ssp_version_t *const *p_version*)

Sets TRNG Framework Code and API version based on compile time macros.

Return values

SSP_SUCCESS	Successful close
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_sf_crypto_trng_api

const sf_crypto_trng_api_t g_sf_crypto_trng_api

Filled in Interface API structure for this Instance.

5.1.3.21 FX_IO Framework

Renesas Synergy Software Package Reference » Framework Layer

FileX adaptation layer for block media device drivers. More...

Data Structures

struct	sf_el_fx_media_partition_data_info_t
struct	sf_el_fx_media_mbr_info_t
struct	sf_el_fx_media_ebr_info_t
struct	sf_el_fx_media_boot_record_table_info_t
struct	sf_el_fx_media_global_open_info_t
struct	sf_el_fx_media_partition_info_t
struct	sf_el_fx_media_info_t
struct	sf_el_fx_callback_args_t
struct	sf_el_fx_config_t
struct	sf_el_fx_instance_ctrl_t
struct	sf_el_fx_t

Macros

#define	SF_EL_FX_API_VERSION_MAJOR (2)
#define	SF_EL_FX_CODE_VERSION_MAJOR (2U)
#define	SF_EL_FX_55AA_SIGNATURE_OFFSET (0x1FEU)
#define	SF_EL_FX_OPEN (0x4649584FU)

Enumerations

enum	sf_el_fx_media_partition_table_update_status_t { SF_EL_FX_PARTITION_TABLE_UPDATE_DISABLE = 0U, SF_EL_FX_PARTITION_TABLE_UPDATE_ENABLE = 1U }
enum	sf_el_fx_media_partition_exist_status_t { SF_EL_FX_NO_PARTITIONS = 0U, SF_EL_FX_MULTIPLE_PARTITIONS = 1U }
enum	sf_el_fx_media_partition_type_t { SF_EL_FX_PARTITION_TYPE_UNKNOWN = 0x00U,

	SF_EL_FX_PARTITION_TYPE_FAT_16_MEMORY_LESS_THAN_32MB = 0x04U, SF_EL_FX_PARTITION_TYPE_EXTENDED = 0x05U, SF_EL_FX_PARTITION_TYPE_FAT_16_MEMORY_MORE_THAN_32MB = 0x06U, SF_EL_FX_PARTITION_TYPE_EXFAT = 0x07U, SF_EL_FX_PARTITION_TYPE_FAT_32 = 0x0BU, SF_EL_FX_PARTITION_TYPE_EXTENDED_INT13 = 0x0FU }
enum	sf_el_fx_media_init_status_t { SF_EL_FX_SYS_UNINIT = 0U, SF_EL_FX_SYS_INIT_PARTIAL = 1U, SF_EL_FX_SYS_INIT_FULL = 2U }
enum	sf_el_fx_media_mbr_ebr_status_t { SF_EL_FX_DONOT_EXIST_OR_INVALID = 0x00U, SF_EL_FX_EXIST_AND_VALID = 0x01U }
enum	sf_el_fx_media_partition_global_open_status_t { SF_EL_FX_PARTITION_GLOBAL_CLOSE = 0U, SF_EL_FX_PARTITION_GLOBAL_OPEN = 1U }
enum	sf_el_fx_media_partition_open_status_t { SF_EL_FX_PARTITION_CLOSE = 0U, SF_EL_FX_PARTITION_OPEN = 1U }
enum	sf_el_fx_media_partition_format_status_t { SF_EL_FX_PARTITION_UNFORMATED = 0U, SF_EL_FX_PARTITION_PRE_RESET_FORMATED = 1U, SF_EL_FX_PARTITION_POST_RESET_FORMATED = 2U }
enum	sf_el_fx_media_partition_ebr_buff_update_t { SF_EL_FX_EBR_BUFF_UPDATE_PRESENT = 0U, SF_EL_FX_EBR_BUFF_UPDATE_NEXT = 1U }

Functions

void	SF_EL_FX_BlockDriver (FX_MEDIA *p_fx_media) Access Block Media device functions open, close, read, write and control. More...

Detailed Description

FileX adaptation layer for block media device drivers.

SF_EL_FX FileX I/O is a single entry function which adapts FileX to Renesas Synergy block media device drivers.

Summary

SF_EL_FX Has no API file.

Macro Definition Documentation

◆ SF_EL_FX_55AA_SIGNATURE_OFFSET

#define SF_EL_FX_55AA_SIGNATURE_OFFSET (0x1FEU)
SSP FileX Support.

◆ SF_EL_FX_API_VERSION_MAJOR

#define SF_EL_FX_API_VERSION_MAJOR (2)
Version of the API defined in this file

◆ SF_EL_FX_CODE_VERSION_MAJOR

#define SF_EL_FX_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ SF_EL_FX_OPEN

#define SF_EL_FX_OPEN (0x4649584FU)
"FIXO" in ASCII. Used to determine if the control block is open.

Enumeration Type Documentation

◆ sf_el_fx_media_init_status_t

enum sf_el_fx_media_init_status_t	
Media initialization status	
Enumerator	
SF_EL_FX_SYS_UNINIT	System not initialized.
SF_EL_FX_SYS_INIT_PARTIAL	System partial initialized.
SF_EL_FX_SYS_INIT_FULL	System full initialized.

◆ sf_el_fx_media_mbr_ebr_status_t

enum sf_el_fx_media_mbr_ebr_status_t	
Media MBR/EBR table status	
Enumerator	
SF_EL_FX_DONOT_EXIST_OR_INVALID	MBR/EBR table do not exist or is invalid.
SF_EL_FX_EXIST_AND_VALID	MBT/EBR table is valid.

◆ sf_el_fx_media_partition_ebr_buff_update_t

enum sf_el_fx_media_partition_ebr_buff_update_t	
Media partition EBR buffer update	
Enumerator	
SF_EL_FX_EBR_BUFF_UPDATE_PRESENT	Update present EBR table.
SF_EL_FX_EBR_BUFF_UPDATE_NEXT	Update next EBR table.

◆ sf_el_fx_media_partition_exist_status_t

enum sf_el_fx_media_partition_exist_status_t	
Media partition exist status	
Enumerator	
SF_EL_FX_NO_PARTITIONS	No partition in memory.
SF_EL_FX_MULTIPLE_PARTITIONS	Multiple partitions in memory.

◆ sf_el_fx_media_partition_format_status_t

enum sf_el_fx_media_partition_format_status_t	
Media partition format status	
Enumerator	
SF_EL_FX_PARTITION_UNFORMATED	Partition is not formated.
SF_EL_FX_PARTITION_PRE_RESET_FORMATED	Partition exist before reset of the system.
SF_EL_FX_PARTITION_POST_RESET_FORMATED	Partition created after system reset.

◆ sf_el_fx_media_partition_global_open_status_t

enum sf_el_fx_media_partition_global_open_status_t	
Media partition global open status	
Enumerator	
SF_EL_FX_PARTITION_GLOBAL_CLOSE	All partitions are closed.
SF_EL_FX_PARTITION_GLOBAL_OPEN	At least one partition is open.

◆ sf_el_fx_media_partition_open_status_t

enum sf_el_fx_media_partition_open_status_t	
Media partition open status	
Enumerator	
SF_EL_FX_PARTITION_CLOSE	Partition is close.
SF_EL_FX_PARTITION_OPEN	Partition is open.

◆ sf_el_fx_media_partition_table_update_status_t

enum sf_el_fx_media_partition_table_update_status_t	
Block Media Control Block Type Media partition table update status	
Enumerator	
SF_EL_FX_PARTITION_TABLE_UPDATE_DISABLE	Partition table update is disabled.
SF_EL_FX_PARTITION_TABLE_UPDATE_ENABLE	Partition table update is enabled.

◆ sf_el_fx_media_partition_type_t

enum sf_el_fx_media_partition_type_t	
Media partition types	
Enumerator	
SF_EL_FX_PARTITION_TYPE_UNKNOWN	Partition type unknown.
SF_EL_FX_PARTITION_TYPE_FAT_16_MEMORY_LESS_THAN_32MB	Partition type fat16 with size less than 32 MB.
SF_EL_FX_PARTITION_TYPE_EXTENDED	Partition type extended.
SF_EL_FX_PARTITION_TYPE_FAT_16_MEMORY_MORE_THAN_32MB	Partition type fat16 with size more than 32 MB.
SF_EL_FX_PARTITION_TYPE_EXFAT	Partition type exFAT.
SF_EL_FX_PARTITION_TYPE_FAT_32	Partition type fat32.
SF_EL_FX_PARTITION_TYPE_EXTENDED_INT13	Partition type extended with interrupt 13.

Function Documentation

◆ SF_EL_FX_BlockDriver()

void SF_EL_FX_BlockDriver (FX_MEDIA * *p_fx_media*)

Access Block Media device functions open, close, read, write and control.

The file system relies on the media to be formatted prior to creating directories and files The sector size and sector count will change depending on the media type and size.

The File Allocation Table (FAT) starts after the reserved sectors in the media. The FAT area is basically an array of 12-bit, 16-bit, or 32-bit entries that determine if that cluster is allocated or part of a chain of clusters comprising a subdirectory or a file. The size of each FAT entry is determined by the number of clusters that need to be represented. If the number of clusters (derived from the total sectors divided by the sectors per cluster) is less than 4,086, 12-bit FAT entries are used. If the total number of clusters is greater than 4,086 and less than or equal to 65,525, 16-bit FAT entries are used. Otherwise, if the total number of clusters is greater than 65,525, 32-bit FAT entries are used. The SF_EL_FX_BlockDriver function is called from the FileX file system driver and issues requests to a Block Media device through the Synergy Block Media Interface. Uses block media driver for accesses.

Parameters

[in,out]	p_fx_media	FileX media control block. All information about each open media device are maintained in by the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide.

Return values

none.	

Note

This function returns nothing, but updates FileX media control block.This function calls sf_el_fx_driver_request_handler

Initialize FileX I/O status to error. It will change to FX_SUCCESS unless an operation fails.

Pass control to sf_el_fx_driver_request_handler, which is responsible for setting FX_MEDIA::fx_media_driver_status in-case of any failure.

- Update FX_MEDIA::fx_media_driver_status to FX_SUCCESS on successful driver request.

sf_el_fx_media_partition_data_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

uint32_t	offset Partition offset.
uint32_t	size Partition size.
sf_el_fx_media_partition_open_status_t	open_status Partition open status.
sf_el_fx_media_partition_format_status_t	format_status Partition format status.

Detailed Description

Individual partition offset, size, open and format status data

---

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_mbr_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_mbr_ebr_status_t	status

	MBR table status.
uint8_t	buff [512] MBR buffer.

Detailed Description

MBR status and buffer data

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_ebr_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_mbr_ebr_status_t	status EBR table status.
uint32_t	base_addr EBR table base address.
uint8_t	buff [512] EBR buffer.

Detailed Description

EBR status, buffer and base address data

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_boot_record_table_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_mbr_info_t	mbr MBR information.
sf_el_fx_media_ebr_info_t	ebr EBR information.

Detailed Description

MBR and EBR table information

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_global_open_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_partition_global_open_status_t	status Global partition open/close status.
uint32_t	counter Partition open counter.

Detailed Description

Global open and counter status of the partition

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_partition_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_partition_exist_status_t	multiple_partitions_status Single or multiple partition status in the memory.
sf_el_fx_media_partition_data_info_t *	p_data Pointer to partition data.
uint32_t	total_count Total number of partitions.
uint32_t	actual_count Actual number of partitions created.

Detailed Description

Multiple partition status with total and actual count of partitions

---

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_media_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_media_init_status_t	init_status Media init status.
sf_el_fx_media_global_open_info_t	global_open Global open status and counter.
sf_el_fx_media_boot_record_table_info_t	boot_record_table Boot table.
sf_el_fx_media_partition_info_t	partition Partition details.
uint32_t	memory_total_sectors Total sectors of the memory.
uint32_t	memory_free_sectors Memory sectors available.

Detailed Description

Available media information

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

uint32_t *	p_hidden_sector Partition base address.
void const *	p_context Placeholder for user data.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_config_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_block_media_instance_t *	p_lower_lvl_block_media Lower level block media pointer.
void const *	p_context Pointer to user-provided context.
void *	p_extend

	Any configuration data needed by the hardware.
uint32_t	total_partitions Total partition as per the user configuration input.
void(*	p_callback)(sf_el_fx_callback_args_t *p_args) User callback to specify partitions offset when partitions exist and board got reset.

Detailed Description

SF_EL_FX configuration block.

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

uint32_t	open Flag to determine if the device is open.
sf_el_fx_media_info_t	media_info Available media and partition details.
sf_block_media_instance_t *	p_lower_lvl_block_media Lower level block media pointer.
void const *	p_context Pointer to user-provided context.

void(*	p_callback)(sf_el_fx_callback_args_t *p_args) User callback to specify partitions offset when partitions exist and board got reset.

Detailed Description

SF_EL_FX instance control block.

The documentation for this struct was generated from the following file:

- sf_el_fx.h

sf_el_fx_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » FX_IO Framework

#include <sf_el_fx.h>

Data Fields

sf_el_fx_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
sf_el_fx_config_t const *	p_config Pointer to the configuration structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- sf_el_fx.h

5.1.3.22 GUIX Synergy Port

Renesas Synergy Software Package Reference » Framework Layer

GUIX adaptation layer. More...

Data Structures

struct	sf_el_gx_instance_ctrl_t

Functions

ssp_err_t	SF_EL_GX_Open (sf_el_gx_ctrl_t *const p_api_ctrl, sf_el_gx_cfg_t const *const p_cfg) GUIX adaptation framework driver for Synergy, open function to configure the framework module. The function initialize RTOS resources used by the module, initialize the control block based on user configuration, and transition the module state to SF_EL_GX_OPENED. This function calls following functions: More...
ssp_err_t	SF_EL_GX_Close (sf_el_gx_ctrl_t *const p_api_ctrl) GUIX adaptation framework for Synergy, Close function. This function calls following functions: More...
ssp_err_t	SF_EL_GX_VersionGet (ssp_version_t *p_version) GUIX adaptation framework for Synergy, Version get function. More...
UINT	SF_EL_GX_Setup (GX_DISPLAY *p_display) GUIX adaptation framework for Synergy, Setup GUIX low level device drivers for Display and D/AVE 2D interface. This function has to be passed to the GUIX Studio display driver setup function gx_studio_display_configure() to let GUIX call this function and configure the GUIX low level device driver(s). This function calls following functions: More...
ssp_err_t	SF_EL_GX_CanvasInit (sf_el_gx_ctrl_t *const p_api_ctrl, GX_WINDOW_ROOT *p_window_root) GUIX adaptation framework for Synergy, Canvas initialization, setup the memory address of first canvas to be rendered. More...

Detailed Description

GUIX adaptation layer.

Function Documentation

◆ SF_EL_GX_CanvasInit()

ssp_err_t SF_EL_GX_CanvasInit (sf_el_gx_ctrl_t *const *p_api_ctrl*, GX_WINDOW_ROOT * *p_window_root*)

GUIX adaptation framework for Synergy, Canvas initialization, setup the memory address of first canvas to be rendered.

Return values

SSP_SUCCESS	Memory address is successfully configured to a canvas.
SSP_ERR_ASSERTION	Invalid control block (NULL pointer) or window root (NULL pointer) passed to driver.
SSP_ERR_INVALID_CALL	Function call was made when the driver is not in SF_EL_GX_CONFIGURED state.
SSP_ERR_INTERNAL	Mutex operation had an error.

Locks the driver to update the context.

Lets GUIX know the first canvas

Unlocks the driver.

◆ SF_EL_GX_Close()

ssp_err_t SF_EL_GX_Close (sf_el_gx_ctrl_t *const *p_api_ctrl*)

GUIX adaptation framework for Synergy, Close function. This function calls following functions:

- tx_mutex_delete() Deletes the mutex for driver
- tx_semaphore_delete() Deletes the semaphore for rendering and displaying synchronization.
- sf_el_gx_d2_close() Finalizes 2D Drawing Engine hardware.
- sf_el_gx_display_close() Finalizes display hardware.

Return values

SSP_SUCCESS	Closed the module successfully.
SSP_ERR_ASSERTION	NULL pointer error happens.
SSP_ERR_NOT_OPEN	SF_EL_GX is not opened.

Note

This function is re-entrant.

Finalizes display hardware

Changes the driver state

Deletes a semaphore for frame buffer flip

Deletes driver global mutex

Clears the temporary storage for the pointer to a control block. This procedure has done in SF_EL_GX_Setup() in the expected function call sequence, but clear it here as well in case SF_EL_GX_Setup() being not called.

◆ SF_EL_GX_Open()

ssp_err_t SF_EL_GX_Open (sf_el_gx_ctrl_t *const *p_api_ctrl*, sf_el_gx_cfg_t const *const *p_cfg*)

GUIX adaptation framework driver for Synergy, open function to configure the framework module. The function initialize RTOS resources used by the module, initialize the control block based on user configuration, and transition the module state to SF_EL_GX_OPENED. This function calls following functions:

- sf_el_gx_open_param_check() Check configuration parameters if parameter check is enabled.
- tx_mutex_create() Creates the mutex for lock the driver during the context update.
- tx_mutex_delete() Deletes the mutex if kernel service calls failed in the process.
- tx_semaphore_create() Creates the semaphore for rendering and displaying synchronization.

Return values

SSP_SUCCESS	Opened the module successfully.
SSP_ERR_ASSERTION	NULL pointer error happened.
SSP_ERR_IN_USE	SF_EL_GX is in-use.
SSP_ERR_INTERNAL	Error happened in kernel service calls.
SSP_ERR_INVALID_ARGUMENT	An invalid argument was passed to the driver.

Note

This function does not initialize the display or rendering hardware but setup function will do.
This function registers a user callback function but it is optional. Set NULL to p_cfg::p_callback if user callback is not required.
The configuration for the frame buffer B (p_cfg::p_framebuffer_b) is optional. Set NULL to p_framebuffer_b in case of a single-buffered system.

Creates global mutex for SF_EL_GX to protect access to the control structure and GUIX low level device drivers setup.

Locks the SF_EL_GX instance until driver setup is done by SF_EL_GX_Setup().

Creates a semaphore for frame buffer flip

Initializes the SF_EL_GX control block

Saves the control block to the global pointer inside the module temporarily. Stored data will be used in sf_el_gx_driver_setup() which will be invoked by GUIX. This pointer will be valid at last but be protected until SF_EL_GX_Setup() is done.

Changes the driver state

◆ SF_EL_GX_Setup()

UINT SF_EL_GX_Setup (GX_DISPLAY * *p_display*)

GUIX adaptation framework for Synergy, Setup GUIX low level device drivers for Display and D/AVE 2D interface. This function has to be passed to the GUIX Studio display driver setup function gx_studio_display_configure() to let GUIX call this function and configure the GUIX low level device driver(s). This function calls following functions:

- tx_mutex_put() Puts the driver global mutex when the low level driver setup is done
- tx_mutex_delete() Deletes the mutex if kernel service calls failed in the process
- _gx_synergy_display_driver_565rgb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is RGB565 format
- _gx_synergy_display_driver_24xrgb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is RGB888, unpacked format
- _gx_display_driver_32argb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is ARGB8888, unpacked format
- sf_el_gx_driver_setup() Setups low level device drivers and overrides the GUIX default callback functions with hardware accelerated functions.

Return values

GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.

Note

Make sure SF_EL_GX_Open() has been called when this function is called back by GUIX. The behavior is not defined if this function were not invoked by GUIX.

Copies the GX_DISPLAY context for later use.

Setups GUIX low level device drivers

Changes the driver state

Clears the temporary storage for the pointer to a control block.

Unlocks the SF_EL_GX instance since driver setup is done

◆ SF_EL_GX_VersionGet()

ssp_err_t SF_EL_GX_VersionGet (ssp_version_t * *p_version*)
GUIX adaptation framework for Synergy, Version get function.

Parameters

[in,out]	p_version	The version number.

Return values

SSP_SUCCESS	Successfully returned the module version.
SSP_ERR_ASSERTION	NULL pointer is passed to function.

Note

This function is re-entrant.

sf_el_gx_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » GUIX Synergy Port

#include <sf_el_gx.h>

Data Fields

GX_DISPLAY *	p_display Pointer to the GUIX display context.
display_instance_t *	p_display_instance Pointer to a display instance.
display_runtime_cfg_t *	p_display_runtime_cfg Pointer to a runtime display configuration.
display_frame_layer_t	inherit_frame_layer Configured Inherit Screen Layer.
void *	p_canvas Pointer to a canvas(reserved)

void *	p_framebuffer_read Pointer to a frame buffer (for displaying)
void *	p_framebuffer_write Pointer to a frame buffer (for rendering)
void(*	p_callback)(sf_el_gx_callback_args_t *p_args) Pointer to callback function.
void *	p_context Pointer to a context.
TX_SEMAPHORE	semaphore Semaphore for the frame buffer flip sync.
bool	rendering_enable Sync flag between Rendering and displaying.
bool	display_list_flushed Flag to show the display list is flushed.
sf_el_gx_state_t	state State of this module.
void *	p_jpegbuffer Pointer to a JPEG work buffer.
uint32_t	jpegbuffer_size Size of a JPEG work buffer.
void *	p_sf_jpeg_decode_instance Pointer to a JPEG framework instance.

_Bool	dave2d_buffer_cache_enabled D/AVE 2D buffer cache enabled/disabled.
uint8_t	bytes_per_pixel Number of bytes per pixel.

Detailed Description

GUIX adaptation layer for SSP. Instance control block for the SSP GUIX adaptation framework

---

The documentation for this struct was generated from the following file:

- sf_el_gx.h

5.1.3.23 EL_LX_NOR

Renesas Synergy Software Package Reference » Framework Layer

LevelX NOR driver implementation. More...

Data Structures

struct	sf_el_lx_nor_memory_settings_t
struct	sf_el_lx_nor_callback_args_t
struct	sf_el_lx_nor_instance_ctrl_t
struct	sf_el_lx_nor_instance_cfg_t

Macros

#define	SF_EL_LX_NOR_API_VERSION_MAJOR (2U)
#define	SF_EL_LX_NOR_CODE_VERSION_MAJOR (2U)

Enumerations

enum	sf_el_lx_nor_event_t { SF_EL_LX_NOR_EVENT_BLOCK_ERASE }

Functions

ssp_err_t	SF_EL_LX_NOR_Open (sf_el_lx_nor_instance_ctrl_t *const p_ctrl, sf_el_lx_nor_instance_cfg_t const *const p_cfg)

	Initializes LevelX NOR frame work read/write and control. More...
ssp_err_t	SF_EL_LX_NOR_Read (sf_el_lx_nor_instance_ctrl_t *const p_ctrl, ULONG *const p_flash, ULONG *const p_dest, ULONG word_count) LevelX NOR driver "read sector" service. More...
ssp_err_t	SF_EL_LX_NOR_Write (sf_el_lx_nor_instance_ctrl_t *const p_ctrl, ULONG *const p_flash, ULONG *const p_src, ULONG word_count) LevelX NOR driver "write sector" service. More...
ssp_err_t	SF_EL_LX_NOR_BlockErase (sf_el_lx_nor_instance_ctrl_t *const p_ctrl, ULONG block, ULONG erase_count) LevelX NOR driver "block erase" service. More...
ssp_err_t	SF_EL_LX_NOR_BlockErasedVerify (sf_el_lx_nor_instance_ctrl_t *const p_ctrl, ULONG block) LevelX NOR driver "block erased verify" service. More...
ssp_err_t	SF_EL_LX_NOR_Close (sf_el_lx_nor_instance_ctrl_t *const p_ctrl) LevelX NOR driver close service. More...

Detailed Description

LevelX NOR driver implementation.

Macro Definition Documentation

◆ SF_EL_LX_NOR_API_VERSION_MAJOR

```
#define SF_EL_LX_NOR_API_VERSION_MAJOR   (2U)
```

Common macro for SSP header files. There is also a corresponding SSP_FOOTER macro at the end of this file. Version of the API defined in this file

◆ SF_EL_LX_NOR_CODE_VERSION_MAJOR

```
#define SF_EL_LX_NOR_CODE_VERSION_MAJOR   (2U)
```

Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sf_el_lx_nor_event_t

enum sf_el_lx_nor_event_t	
Options for the callback events.	
Enumerator	
SF_EL_LX_NOR_EVENT_BLOCK_ERASE	Block erase event triggered.

Function Documentation

◆ SF_EL_LX_NOR_BlockErase()

ssp_err_t SF_EL_LX_NOR_BlockErase (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*, ULONG *block*, ULONG *erase_count*)

LevelX NOR driver "block erase" service.

This is responsible for erasing the specified block of the NOR flash.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR framework instance.
[in]	block	Specifies which NOR block to erase.
[in]	erase_count	Provided for diagnostic purposes(currently unused).

Return values

SSP_SUCCESS	LevelX NOR flash block erase successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for erasing.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:erase

Validate the parameters

If the driver is not open return an error.

Calculate the block address

Erase using underlying API

Call the user function if available

Prepare the callback arguments

Invoke callback function

◆ SF_EL_LX_NOR_BlockErasedVerify()

ssp_err_t SF_EL_LX_NOR_BlockErasedVerify (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*, ULONG *block*)

LevelX NOR driver "block erased verify" service.

This is responsible for verifying the specified block of the NOR flash is erased.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR framework instance.
[in]	block	Specifies which block to verify that it is erased.

Return values

SSP_SUCCESS	LevelX flash block erase verification successful.
SSP_ERR_ASSERTION	p_ctrl or lower level driver is NULL.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for verifying.
SSP_ERR_NOT_ERASED	The block is not erased properly.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:read

Validate the parameters

Check whether the driver is in OPEN state

Loop to check if the block is erased.

Check whether the driver read is success or not

Iterate over buffer and validate

Is this word erased?

◆ SF_EL_LX_NOR_Close()

ssp_err_t SF_EL_LX_NOR_Close (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*)

LevelX NOR driver close service.

This is responsible for closing the driver properly.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR framework instance.

Return values

SSP_SUCCESS	LevelX flash is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for closing.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:close

Validate the parameters

Check whether the driver is in OPEN state

Close underlying API

Reset OPEN state

◆ SF_EL_LX_NOR_Open()

ssp_err_t SF_EL_LX_NOR_Open (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*,
sf_el_lx_nor_instance_cfg_t const *const *p_cfg*)

Initializes LevelX NOR frame work read/write and control.

Name of module used by error logger macro Calls lower level driver initialization function.

Parameters

[in,out]	p_ctrl	Control block for the LevelX NOR framework instance.
[in,out]	p_cfg	LevelX NOR driver instance.

Return values

SSP_SUCCESS	LevelX NOR driver is successfully opened.
SSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.
SSP_ERR_ALREADY_OPEN	Driver is already in OPEN state.
SSP_ERR_INVALID_ARGUMENT	p_memory_settings structure configured to invalid values.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:open
- sf_memory_api_t:infoGet
- sf_memory_api_t:close

Validate the parameters

Check whether instance is already open

Update the instance control block

Open the underlying memory instance

Get the underlying NOR flash info

If unable to get NOR flash info, close lower layer driver

Update memory region info

Setup the base address of the flash memory.

Setup geometry of the flash.

Setup the base address of the flash memory.

Setup geometry of the flash.

Mark control block open so subsequent calls know the device is open.

◆ SF_EL_LX_NOR_Read()

ssp_err_t SF_EL_LX_NOR_Read (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash*, ULONG *const *p_dest*, ULONG *word_count*)

LevelX NOR driver "read sector" service.

This is responsible for reading a specific sector in a specific block of the NOR flash. All error checking and correcting logic is the responsibility of the this service.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR framework instance.
[in]	p_flash	Specifies the address of a logical sector within a NOR flash block of memory.
[in,out]	p_dest	Specifies where to place the sector contents.
[in]	word_count	Specifies how many 32-bit words to read.

Return values

SSP_SUCCESS	LevelX NOR flash sector read successful.
SSP_ERR_ASSERTION	p_ctrl, p_flash or p_dest is NULL.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for reading.
SSP_ERR_INVALID_ARGUMENT	Requested range can't fit in the flash address range.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:read

Validate the parameters

Check whether the driver is in OPEN state

Read from underlying API

◆ SF_EL_LX_NOR_Write()

ssp_err_t SF_EL_LX_NOR_Write (sf_el_lx_nor_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash*, ULONG *const *p_src*, ULONG *word_count*)

LevelX NOR driver "write sector" service.

This is responsible for writing a specific sector into a block of the NOR flash. All error checking is the responsibility of the this service.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR framework instance.
[in,out]	p_flash	Specifies the address of a logical sector within a NOR flash block of memory.
[in]	p_src	Specifies the source of the write.
[in]	word_count	Specifies how many 32-bit words to write.

Return values

SSP_SUCCESS	LevelX NOR flash sector write successful.
SSP_ERR_ASSERTION	p_ctrl, p_flash or p_src is NULL.
SSP_ERR_NOT_OPEN	Driver not in OPEN state for writing.
SSP_ERR_INVALID_ARGUMENT	Requested range can't fit in the flash address range.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- sf_memory_api_t:write

Validate the parameters

Check whether the driver is in OPEN state

Write to underlying API

sf_el_lx_nor_memory_settings_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » EL_LX_NOR

#include <sf_el_lx_nor.h>

Data Fields

uint32_t	absolute_start_addr Starting address of memory partition.
uint32_t	size Size of the partitioned region.

Detailed Description

SF_EL_LX_NOR memory settings for partition functionality.

The documentation for this struct was generated from the following file:

- sf_el_lx_nor.h

sf_el_lx_nor_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » EL_LX_NOR

#include <sf_el_lx_nor.h>

Data Fields

sf_el_lx_nor_event_t	event LevelX NOR driver callback event.
void const *	p_context Placeholder for user data.
uint32_t	erase_block_number Erase block number.
uint32_t	erase_block_count Erase count of specified block number.

Detailed Description

SF_EL_LX_NOR callback arguments definitions

The documentation for this struct was generated from the following file:

- sf_el_lx_nor.h

sf_el_lx_nor_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » EL_LX_NOR

#include <sf_el_lx_nor.h>

Data Fields

sf_memory_instance_t const *	p_lower_lvl Lower level memory pointer.
LX_NOR_FLASH *	p_lx_nor_flash Pointer to the LevelX nor flash instance.
sf_memory_region_info_t *	p_region_info Memory region information.
void const *	p_context Placeholder for user data. Passed to the user callback.
sf_el_lx_nor_memory_settings_t const *	p_memory_settings Pointer to memory settings structure for partitioning functionality.
void(*	p_callback)(sf_el_lx_nor_callback_args_t *p_args) Callback function.
uint32_t	open Used to determine if framework is initialized.

Detailed Description

SF_EL_LX_NOR Control Block Type

The documentation for this struct was generated from the following file:

- sf_el_lx_nor.h

sf_el_lx_nor_instance_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » EL_LX_NOR

#include <sf_el_lx_nor.h>

Data Fields

sf_memory_instance_t const *	p_lower_lvl Lower level memory pointer.
LX_NOR_FLASH *	p_lx_nor_flash Pointer to the LevelX nor flash instance.
void const *	p_context Placeholder for user data. Passed to the user callback.
sf_el_lx_nor_memory_settings_t const *	p_memory_settings Pointer to memory settings structure for partitioning functionality.
void(*	p_callback)(sf_el_lx_nor_callback_args_t *p_args) Callback function.

Detailed Description

SF_EL_LX_NOR Config Block Type

The documentation for this struct was generated from the following file:

- sf_el_lx_nor.h

5.1.3.24 USB Communication Framework V2

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated USBX CDC ACM device implementation. More...

Data Structures

struct	sf_el_ux_comms_instance_ctrl_t

Functions

ssp_err_t	SF_EL_UX_COMMS_Open (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_cfg_t const *const p_cfg) Initializes a USB channel for CDC ACM mode. More...
ssp_err_t	SF_EL_UX_COMMS_Close (sf_comms_ctrl_t *const p_api_ctrl) Releases all the ThreadX Resources. More...
ssp_err_t	SF_EL_UX_COMMS_Read (sf_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout) Read data from the USBX CDC-ACM driver. More...
ssp_err_t	SF_EL_UX_COMMS_Write (sf_comms_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout) Write data to the USBX CDC-ACM framework. More...
ssp_err_t	SF_EL_UX_COMMS_Lock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type, UINT timeout) Lock the USB COM resource. More...
ssp_err_t	SF_EL_UX_COMMS_Unlock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type) Unlock the USB COM resource. More...

ssp_err_t	SF_EL_UX_COMMS_VersionGet (ssp_version_t *const p_version) Get driver version. More...

Detailed Description

RTOS-integrated USBX CDC ACM device implementation.

Function Documentation

◆ SF_EL_UX_COMMS_Close()

ssp_err_t SF_EL_UX_COMMS_Close (sf_comms_ctrl_t *const *p_api_ctrl*)

Releases all the ThreadX Resources.

Return values

SSP_SUCCESS	Channel successfully closed
SSP_ERR_ASSERTION	Pointer to control block is NULL
SSP_ERR_NOT_OPEN	Module is not opened.

Note

This function is reentrant.

Check if module has been opened.

Delete transmit mutex.

Delete receive mutex.

Deletes a semaphore for USBX CDC instance

◆ SF_EL_UX_COMMS_Lock()

ssp_err_t SF_EL_UX_COMMS_Lock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*, UINT *timeout*)

Lock the USB COM resource.

Return values

SSP_SUCCESS	Locking a USB COM resource successful.
SSP_ERR_ASSERTION	Pointer to control block is NULL.
SSP_ERR_NOT_OPEN	Module is not opened.
SSP_ERR_TIMEOUT	Mutex not available in timeout.

If lock type is ALL, both TX and RX gets locked else either TX or RX is locked.

If TX or RX fails to acquire mutex, return error.

◆ SF_EL_UX_COMMS_Open()

ssp_err_t SF_EL_UX_COMMS_Open (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_cfg_t const *const *p_cfg*)

Initializes a USB channel for CDC ACM mode.

Parameters

[in]	p_api_ctrl	Pointer to control structure block.
[in]	p_cfg	Pointer to configuration structure block. This parameter is not used in the framework hence the NULL parameter check not implemented.

Return values

SSP_SUCCESS	Channel opened successfully.
SSP_ERR_ASSERTION	p_api_ctrl pointer parameter to control block is NULL.
SSP_ERR_TIMEOUT	Semaphore not available in timeout.
SSP_ERR_INTERNAL	Transmit/Receive mutex or Semaphore creation fails.
SSP_ERR_IN_USE	Channel/peripheral is running/busy.

Note

This function is reentrant.

Create semaphore if the USBX CDC instance is not ready.

If the semaphore creation fails, return error.

Suspend here until a USBX CDC instance is created by the USBX CDC for this module.

Create the mutex for protecting the access to control structure for transmit and related hardware

Create the mutex for protecting the access to control structure for receive and related hardware

Mark control block open.

◆ SF_EL_UX_COMMS_Read()

ssp_err_t SF_EL_UX_COMMS_Read (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, UINT const *timeout*)

Read data from the USBX CDC-ACM driver.

Return values

SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_INTERNAL	An error has occurred if usb read operation fails or buffer overflow occurred.
SSP_ERR_TIMEOUT	Receive mutex get timed out

Note

This API is reentrant.

Get mutex.

Set timeout value in the transfer request.

If there is data leftover from the last packet, use it.

Read from the CDC class.

Release mutex in case of buffer overflow or read error .

Release mutex for Read API.

◆ SF_EL_UX_COMMS_Unlock()

ssp_err_t SF_EL_UX_COMMS_Unlock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*)

Unlock the USB COM resource.

Return values

SSP_SUCCESS	Unlocking a USB COM resource successful.
SSP_ERR_ASSERTION	Pointer to control block is NULL.
SSP_ERR_NOT_OPEN	Module is not opened.
SSP_ERR_INTERNAL	Failed to release the mutex.

Unlock the USB CDC COM resource as per the user request.

If USB CDC COM resource fails to unlock,returns error.

◆ SF_EL_UX_COMMS_VersionGet()

ssp_err_t SF_EL_UX_COMMS_VersionGet (ssp_version_t *const *p_version*)

Get driver version.

Parameters

[out]	p_version	Version will be stored here.

Note

This function is reentrant.

◆ SF_EL_UX_COMMS_Write()

ssp_err_t SF_EL_UX_COMMS_Write (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, UINT const *timeout*)

Write data to the USBX CDC-ACM framework.

Return values

SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_INTERNAL	An error has occurred, when usb write operation fails.
SSP_ERR_TIMEOUT	Transmit mutex get timed out or when DTR and RTS state setting gets timed out.

Note

This function is reentrant.

Wait for DTR and RTS state to set.

Get Transmit mutex.

Set timeout value in the transfer request.

Release mutex in case of error condition and log the error .

Release Transmit mutex for Write API.

sf_el_ux_comms_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USB Communication Framework V2

#include <sf_el_ux_comms_v2.h>

Detailed Description

USBX CDC ACM device communications instance control structure. DO NOT INITIALIZE. Initialization occurs when sf_comms_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_el_ux_comms_v2.h

5.1.3.25 External IRQ Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated external IRQ Framework. More...

Data Structures

struct	sf_external_irq_instance_ctrl_t

Functions

ssp_err_t	SF_EXTERNAL_IRQ_Open (sf_external_irq_ctrl_t *const p_api_ctrl, sf_external_irq_cfg_t const *const p_cfg) Configure external IRQ and optionally enable external IRQ callbacks. Implements sf_external_irq_api_t::open. More...
ssp_err_t	SF_EXTERNAL_IRQ_Wait (sf_external_irq_ctrl_t *const p_api_ctrl, ULONG const timeout) Get semaphore with specified timeout for external interrupt to expire. Implements sf_external_irq_api_t::wait. More...
ssp_err_t	SF_EXTERNAL_IRQ_VersionGet (ssp_version_t *const p_version) Get version and store it in provided pointer p_version. Implements sf_external_irq_api_t::versionGet. More...
ssp_err_t	SF_EXTERNAL_IRQ_Close (sf_external_irq_ctrl_t *const p_api_ctrl) Close channel at HAL layer and delete the semaphore . Implements sf_external_irq_api_t::close. More...

Detailed Description

RTOS-integrated external IRQ Framework.

Summary

This module is a ThreadX-aware external IRQ Framework for external inputs such as switches or other binary signals.

Function Documentation

◆ SF_EXTERNAL_IRQ_Close()

ssp_err_t SF_EXTERNAL_IRQ_Close (sf_external_irq_ctrl_t *const *p_api_ctrl*)

Close channel at HAL layer and delete the semaphore . Implements sf_external_irq_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNSUPPORTED	Unsupported operation.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- external_irq_api_t::close

Close low level driver

Clear information from control block so other functions know this instance is closed

Delete the semaphore used

◆ SF_EXTERNAL_IRQ_Open()

ssp_err_t SF_EXTERNAL_IRQ_Open (sf_external_irq_ctrl_t *const *p_api_ctrl*, sf_external_irq_cfg_t const *const *p_cfg*)

Configure external IRQ and optionally enable external IRQ callbacks. Implements sf_external_irq_api_t::open.

The SF_EXTERNAL_IRQ_Open() function creates semaphore for the external IRQ channel used, then calls the HAL driver open function. After successful initialization, the external IRQ is ready for use.

Return values

SSP_SUCCESS	Initialization was successful and external interrupt has started.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg, p_api, or p_api->open. See HAL driver for other possible causes.
SSP_ERR_IN_USE	This channel is already open.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- external_irq_api_t::open

Note

This function is reentrant for any channel.

Save driver structure for use in other framework layer functions

Create semaphore for use with wait function

Prepare configuration for lower layer

Open lower layer

If low level initialization failed, delete the semaphore and exit the function with the error code

Delete the semaphore.

log the error and return the error

Mark control block open so other tasks know it is valid

◆ SF_EXTERNAL_IRQ_VersionGet()

ssp_err_t SF_EXTERNAL_IRQ_VersionGet (ssp_version_t *const *p_version*)

Get version and store it in provided pointer p_version. Implements sf_external_irq_api_t::versionGet .

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

◆ SF_EXTERNAL_IRQ_Wait()

ssp_err_t SF_EXTERNAL_IRQ_Wait (sf_external_irq_ctrl_t *const *p_api_ctrl*, ULONG const *timeout*)

Get semaphore with specified timeout for external interrupt to expire. Implements sf_external_irq_api_t::wait.

Return values

SSP_SUCCESS	External interrupt stopped successfully.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_EXTERNAL_IRQ_Open to configure.
SSP_ERR_TIMEOUT	Time out happens while waiting a semaphore.
SSP_ERR_WAIT_ABORTED	Suspension was aborted by another thread.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Wait for semaphore post from ISR

sf_external_irq_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » External IRQ Framework

#include <sf_external_irq.h>

Data Fields

uint32_t	open Used by driver to check if control block is valid.

TX_SEMAPHORE	semaphore Semaphore used for SF_EXTERNAL_IRQ_Wait.
external_irq_instance_t const *	p_lower_lvl_irq Pointer to lower level driver instance.
bool	callback_used Used by driver to check if wait can be used.

Detailed Description

Instance control block. DO NOT INITIALIZE. Initialization occurs when sf_external_irq_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_external_irq.h

5.1.3.26 I2C Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated I2C Framework. More...

Data Structures

struct	sf_i2c_instance_ctrl_t

Functions

ssp_err_t	SF_I2C_Open (sf_i2c_ctrl_t *const p_api_ctrl, sf_i2c_cfg_t const *const p_cfg) Initialize a I2C bus and open low level I2C driver. More...
ssp_err_t	SF_I2C_Read (sf_i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout) Start the transfer process and receive data from I2C device. More...

ssp_err_t	SF_I2C_Write (sf_i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout) Start the transfer process and send data on I2C bus. More...
ssp_err_t	SF_I2C_Reset (sf_i2c_ctrl_t *const p_api_ctrl, uint32_t const timeout) Abort any in-progress transfer. More...
ssp_err_t	SF_I2C_Close (sf_i2c_ctrl_t *const p_api_ctrl) Close the I2C device designated by the control handle and close the RTOS services used by the bus if last device is connected to the bus calls this API, else decrement the device count. More...
ssp_err_t	SF_I2C_Lock (sf_i2c_ctrl_t *const p_api_ctrl) Lock the bus for a device. Once bus is locked by a device it can not be used by other devices. More...
ssp_err_t	SF_I2C_Unlock (sf_i2c_ctrl_t *const p_api_ctrl) Unlock the locked bus and make the bus usable for other devices. More...
ssp_err_t	SF_I2C_VersionGet (ssp_version_t *const p_version) Get the version information of the framework. More...
ssp_err_t	SF_I2C_LockWait (sf_i2c_ctrl_t *const p_api_ctrl, uint32_t const timeout) Lock the I2C Bus resource. Once bus is locked by a device it can not be used by other devices. More...

Detailed Description

RTOS-integrated I2C Framework.

SSP I2C framework driver API

.

Summary

This is a ThreadX-aware I2C driver API. The API implements the I2C Framework interface and can

access several hardware peripherals at the HAL layer. The connection to the HAL layer is established by passing in a driver structure in SF_I2C_Open.

Interface Used

See also
I2C Framework

Function Documentation

◆ SF_I2C_Close()

ssp_err_t SF_I2C_Close (sf_i2c_ctrl_t *const *p_api_ctrl*)

Close the I2C device designated by the control handle and close the RTOS services used by the bus if last device is connected to the bus calls this API, else decrement the device count.

Return values

SSP_SUCCESS	Device is successfully closed.
SSP_ERR_NOT_OPEN	Device was not even opened.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:
- i2c_api_master_t::close

Note

This function is reentrant for any device.

Check whether device is opened or not.

Acquire the device count mutex before accessing the shared resource in close.

Check the count of opened devices on the bus. If there are no devices opened or all other devices on the bus are closed then close the low level I2C driver and release the RTOS services used by the bus.

Get the low level control in use.

Close low level driver.

Delete RTOS services used by the bus.

Decrement device count.

Delete the device count mutex

Decrement device count.

Release the device count mutex

Set device to closed state and restarted flag to false.

◆ SF_I2C_Lock()

ssp_err_t SF_I2C_Lock (sf_i2c_ctrl_t *const *p_api_ctrl*)

Lock the bus for a device. Once bus is locked by a device it can not be used by other devices.

Return values

SSP_SUCCESS	I2C channel is successfully locked.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

Note

This function is reentrant for any device.

Check whether device is opened or not.

Get mutex since this will access hardware registers.

Reconfigure the device address, if necessary

◆ SF_I2C_LockWait()

ssp_err_t SF_I2C_LockWait (sf_i2c_ctrl_t *const *p_api_ctrl*, uint32_t const *timeout*)

Lock the I2C Bus resource. Once bus is locked by a device it can not be used by other devices.

Return values

SSP_SUCCESS	I2C channel is successfully locked within the specified timeout.
SSP_ERR_ASSERTION	Pointer to I2C control block is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_TIMEOUT	Mutex not available in timeout.

Note

This function is reentrant for any device.

Check whether device is opened or not.

Get the mutex for this device.

Reconfigure the device address, if necessary

◆ SF_I2C_Open()

ssp_err_t SF_I2C_Open (sf_i2c_ctrl_t *const *p_api_ctrl*, sf_i2c_cfg_t const *const *p_cfg*)

Initialize a I2C bus and open low level I2C driver.

Return values

SSP_SUCCESS	I2C device is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_cfg, Pointer to Open, Close, Read, Write, or reset API interfaces,p_cfg->p_bus.
SSP_ERR_INTERNAL	Internal error occurred.
SSP_ERR_ALREADY_OPEN	Same I2C framework device is already open.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is

- i2c_api_master_t::open

Note

This function is reentrant for any channel.
Control handle must be cleared by caller before calling this function.

Check whether device is already opened or not.

Copy bus pointer to control

Set framework level callback function.

Save context for use in ISRs.

Enter a critical section before checking the device count mutex status.

Check if device count mutex is already created. If not then create the mutex.

Create device_count_mutex. This is used to protect shared variable device_count in bus control structure.

If mutex create fails, return error.

Exit critical section

Acquire the device count mutex before accessing the shared resource. Try again if the mutex was deleted in close.

Increment device count.

Release the device count mutex

Save device configuration for reconfiguration.

Set device state as Opened.

Initialize restarted flag to false.

◆ SF_I2C_Read()

ssp_err_t SF_I2C_Read (sf_i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*, uint32_t const *timeout*)

Start the transfer process and receive data from I2C device.

Return values

SSP_SUCCESS	Data received successfully.
SSP_ERR_NOT_OPEN	Device instance not opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_dest, bytes, timeout.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- i2c_api_master_t::read

Check whether device is opened or not.

Start transfer process - check reconfiguration, get Mutex.

Get the low level control in use.

Perform read.

Wait for callback to set event flag.

Finish transfer.

◆ SF_I2C_Reset()

ssp_err_t SF_I2C_Reset (sf_i2c_ctrl_t *const *p_api_ctrl*, uint32_t const *timeout*)

Abort any in-progress transfer.

Return values

SSP_SUCCESS	Channel was reseted without issue.
SSP_ERR_NOT_OPEN	Device was not even opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_INTERNAL	Internal error occurred.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- i2c_api_master_t::reset

Check whether device is opened or not.

Get the low level control in use.

Get mutex since this will access hardware registers.

◆ SF_I2C_Unlock()

ssp_err_t SF_I2C_Unlock (sf_i2c_ctrl_t *const *p_api_ctrl*)

Unlock the locked bus and make the bus usable for other devices.

Return values

SSP_SUCCESS	I2C bus is successfully unlocked.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

Note

This function is reentrant for any device.

Check whether device is opened or not.

Release the mutex so that others can use the bus.

◆ SF_I2C_VersionGet()

ssp_err_t SF_I2C_VersionGet (ssp_version_t *const *p_version*)

Get the version information of the framework.

Return values

SSP_SUCCESS	Got version number successfully.
SSP_ERR_ASSERTION	Following parameters is NULL: p_version.

Checks error. Further parameter checking can be done at the driver layer.

◆ SF_I2C_Write()

ssp_err_t SF_I2C_Write (sf_i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*, uint32_t const *timeout*)

Start the transfer process and send data on I2C bus.

Return values

SSP_SUCCESS	Data written successfully.
SSP_ERR_NOT_OPEN	Device instance not opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_src, bytes.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- i2c_api_master_t::write

Check whether device is opened or not.

Start transfer process - check reconfiguration, get Mutex.

Get the low level control in use.

Perform write.

Wait for callback to set event flag.

Finish transfer.

sf_i2c_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » I2C Framework

#include <sf_i2c.h>

Data Fields

sf_i2c_bus_t *	p_bus Bus using this device. Copy from configuration structure.
i2c_master_instance_t const *	p_lower_lvl_i2c I2C instance.
i2c_cfg_t	lower_lvl_cfg Used to reconfigure I2C driver.
i2c_ctrl_t *	p_lower_lvl_ctrl I2C peripheral control block.
sf_i2c_dev_state_t	dev_state Device status.
bool	restarted Indicates whether device issued a restart.

Detailed Description

I2C instance control block. DO NOT INITIALIZE. Initialization occurs when sf_i2c_api_t::open is called.

The documentation for this struct was generated from the following file:

- sf_i2c.h

5.1.3.27 JPEG Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated JPEG Framework. More...

Data Structures

struct	sf_jpeg_decode_instance_ctrl_t

Macros

#define	SF_JPEG_DECODE_CODE_VERSION_MAJOR (2U)
#define	SF_JPEG_DECODE_OPEN (0x4A504547U)
#define	SF_JPEG_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &s_sf_jpeg_version)

Functions

ssp_err_t	sf_jpeg_initialize (sf_jpeg_decode_instance_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg) Acquires mutex, then handles driver initialization at the HAL layer. This function releases the mutex before returns to the caller. More...
ssp_err_t	SF_JPEG_Decode_Open (sf_jpeg_decode_ctrl_t *const p_api_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg) Parameter checking and initialize JPEG decode with sf_jpeg_initialize helper function and marking the open flag in control block. More...
ssp_err_t	SF_JPEG_Decode_InputBufferSet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const buffer_size) Configures JPEG coded input data. More...
ssp_err_t	SF_JPEG_Decode_LinesDecodedGet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, uint32_t *const p_lines) Obtain number of lines decoded by the codec. More...
ssp_err_t	SF_JPEG_Decode_HorizontalStrideSet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, uint32_t horizontal_stride) Configure the horizontal stride value. More...
ssp_err_t	SF_JPEG_Decode_ImageSubsampleSet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image at runtime. More...

ssp_err_t	SF_JPEG_Decode_OutputBufferSet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, void *p_buffer, uint32_t buffer_size) Configure the decode output buffer. More...
ssp_err_t	SF_JPEG_Decode_Wait (sf_jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout) Wait for current JPEG codec operation to finish. More...
ssp_err_t	SF_JPEG_Decode_StatusGet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_status_t *const p_status) Obtain JPEG codec status. This function can be used to poll the device instead of using SF_JPEG_Decode_Wait() to block on JPEG operations. More...
ssp_err_t	SF_JPEG_Decode_PixelFormatGet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_color_space_t *const p_color_space) Obtain the format of the image. This function is only useful for decoding a JPEG image. More...
ssp_err_t	SF_JPEG_Decode_ImageSizeGet (sf_jpeg_decode_ctrl_t *const p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) Obtain the size of the image. This function is only useful for decoding a JPEG image. More...
ssp_err_t	SF_JPEG_Decode_Close (sf_jpeg_decode_ctrl_t *const p_api_ctrl) Close JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded. More...
ssp_err_t	SF_JPEG_Decode_VersionGet (ssp_version_t *const p_version) Get version and store it in provided pointer p_version. More...

Detailed Description

RTOS-integrated JPEG Framework.

Macro Definition Documentation

◆ SF_JPEG_DECODE_CODE_VERSION_MAJOR

#define SF_JPEG_DECODE_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ SF_JPEG_DECODE_OPEN

#define SF_JPEG_DECODE_OPEN (0x4A504547U)
"JPEG" in ASCII, used to identify general JPEG control block

◆ SF_JPEG_ERROR_RETURN

#define SF_JPEG_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &s_sf_jpeg_version)
Macro for error logger.

Function Documentation

◆ SF_JPEG_Decode_Close()

ssp_err_t SF_JPEG_Decode_Close (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*)

Close JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded.

Precondition

Call SF_JPEG_Decode_Open() to configure the timer before using this function.

Return values

SSP_SUCCESS	The JPEG decode device is successfully closed.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.

Call the low level driver to close the JPEG device.

Clear information from control block so other functions know this block is closed

◆ SF_JPEG_Decode_HorizontalStrideSet()

ssp_err_t SF_JPEG_Decode_HorizontalStrideSet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, uint32_t *horizontal_stride*)

Configure the horizontal stride value.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	Horizontal Stride value is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::horizontalStrideSet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_ImageSizeGet()

ssp_err_t SF_JPEG_Decode_ImageSizeGet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, uint16_t * *p_horizontal_size*, uint16_t * *p_vertical_size*)

Obtain the size of the image. This function is only useful for decoding a JPEG image.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	The JPEG image size is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::imageSizeGet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_ImageSubsampleSet()

ssp_err_t SF_JPEG_Decode_ImageSubsampleSet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_subsample_t *horizontal_subsample*, jpeg_decode_subsample_t *vertical_subsample*)

Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image at runtime.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	Image subsample values are successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::imageSubsampleSet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_InputBufferSet()

ssp_err_t SF_JPEG_Decode_InputBufferSet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, void *const *p_buffer*, uint32_t const *buffer_size*)

Configures JPEG coded input data.

This API configures the decode input buffer address register. After the input buffer address is set, the driver checks whether the output buffer address is set, and verifies that the output buffer size is large enough to hold at least eight output lines of data. If both the input buffer and output buffer are set properly, the driver automatically starts the decode process.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	Decode input buffer is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::inputBufferSet

Obtain mutex before making HAL-level driver call.

Call the HAL driver layer inputBufferSet routine.

◆ SF_JPEG_Decode_LinesDecodedGet()

ssp_err_t SF_JPEG_Decode_LinesDecodedGet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_lines*)

Obtain number of lines decoded by the codec.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	Lines decoded value is successfully obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::linesDecodedGet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_Open()

ssp_err_t SF_JPEG_Decode_Open (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, sf_jpeg_decode_cfg_t const *const *p_cfg*)

Parameter checking and initialize JPEG decode with sf_jpeg_initialize helper function and marking the open flag in control block.

Return values

SSP_SUCCESS	JPEG Decode framework is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_ctrl or p_cfg.
SSP_ERR_ALREADY_OPEN	JPEG Decode framework is already open.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Initialize the JPEG framework

Save driver structure pointer for use in other framework layer functions.

Mark control block open so subsequent calls know the device is open.

◆ SF_JPEG_Decode_OutputBufferSet()

ssp_err_t SF_JPEG_Decode_OutputBufferSet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, void * *p_buffer*, uint32_t *buffer_size*)

Configure the decode output buffer.

This API configures the decode output buffer address register. After the output buffer address is set, the driver computers the number of output lines the buffer is able to hold. The hardware requires the number out output lines to decode at a time is multiple of eight. If both the input buffer and output buffer are set properly, the driver automatically starts the decode process.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	Output buffer is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::outputBufferSet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_PixelFormatGet()

ssp_err_t SF_JPEG_Decode_PixelFormatGet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_color_space_t *const *p_color_space*)

Obtain the format of the image. This function is only useful for decoding a JPEG image.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	The JPEG image size is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::pixelFormatGet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_StatusGet()

ssp_err_t SF_JPEG_Decode_StatusGet (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_status_t *const *p_status*)

Obtain JPEG codec status. This function can be used to poll the device instead of using SF_JPEG_Decode_Wait() to block on JPEG operations.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	The JPEG status information is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::statusGet

Obtain mutex before making HAL-level driver call.

◆ SF_JPEG_Decode_VersionGet()

ssp_err_t SF_JPEG_Decode_VersionGet (ssp_version_t *const *p_version*)

Get version and store it in provided pointer p_version.

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version is null.

◆ SF_JPEG_Decode_Wait()

ssp_err_t SF_JPEG_Decode_Wait (sf_jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_status_t *const *p_status*, uint32_t *timeout*)

Wait for current JPEG codec operation to finish.

Precondition

Call SF_JPEG_Decode_Open() to configure the JPEG codec block before using this function.

Return values

SSP_SUCCESS	The wait function returns successfully.
SSP_ERR_ASSERTION	p_ctrl or p_status is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_TIMEOUT	The wait operation timed out, the underlying driver did not response in time.
SSP_ERR_WAIT_ABORTED	System internal error occurred.

Obtain mutex before making HAL-level driver call.

◆ sf_jpeg_initialize()

ssp_err_t sf_jpeg_initialize (sf_jpeg_decode_instance_ctrl_t *const *p_ctrl*, sf_jpeg_decode_cfg_t const *const *p_cfg*)

Acquires mutex, then handles driver initialization at the HAL layer. This function releases the mutex before returns to the caller.

Parameters

[in,out]	p_ctrl	Control handle for JPEG framework context for a device.
[in]	p_cfg	Pointer to JPEG framework Configuration Structure.

Return values

SSP_SUCCESS	JPEG Decode driver is successfully opened.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- jpeg_decode_api_t::open

Create event flags for use with wait function

Create the mutex for this instance, this mutex is used to protect access to lower level hardware

Duplicate the content of p_cfg.

Install framework callback and context.

Call the low level driver to configure the JPEG device.

If the operation failed, delete the resources.

sf_jpeg_decode_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » JPEG Framework

#include <sf_jpeg_decode.h>

Data Fields

uint32_t	open Indicate whether the driver is open.

uint32_t	state Used by driver to check if pointer to control block is valid.
TX_MUTEX	mutex Mutex used to protect access to lower level driver hardware.
TX_EVENT_FLAGS_GROUP	events Event flags used by the HAL driver to notify the framework driver of.
jpeg_decode_instance_t const *	p_lower_lvl_jpeg_decode Pointer to lower level instance.

Detailed Description

JPEG framework instance control block. DO NOT INITIALIZE. Initialization occurs when sf_jpeg_decode_api_t::open is called.

The documentation for this struct was generated from the following file:

- sf_jpeg_decode.h

5.1.3.28 Memory framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Memory framework for QSPI NOR driver. More...

Data Structures

struct	sf_memory_qspi_nor_instance_ctrl_t
struct	sf_memory_qspi_nor_cfg_t

Macros

#define	SF_MEMORY_QSPI_NOR_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_MEMORY_QSPI_NOR_Open (sf_memory_ctrl_t *const p_ctrl,

	sf_memory_cfg_t const *const p_cfg) Open the SF Memory QSPI Nor driver module. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_Close (sf_memory_ctrl_t *const p_ctrl) Close the Memory QSPI NOR driver module. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_Read (sf_memory_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const memory_address, uint32_t const num_bytes) Read data from the flash. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_Write (sf_memory_ctrl_t *const p_ctrl, uint8_t *const p_src_address, uint32_t const memory_address, uint32_t const num_bytes) Program data to the flash. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_Flush (sf_memory_ctrl_t *const p_ctrl) Flush any pending data to the disk. This is not required for QSPI NOR Flash. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_Erase (sf_memory_ctrl_t *p_ctrl, uint32_t const memory_address, uint32_t const num_bytes) Erase a number of bytes from the flash. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_InfoGet (sf_memory_ctrl_t *const p_ctrl, sf_memory_info_t *const p_info) Returns the information about the flash. More...
ssp_err_t	SF_MEMORY_QSPI_NOR_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...

Detailed Description

RTOS-integrated Memory framework for QSPI NOR driver.

Name of module used by error logger macro

Macro Definition Documentation

◆ SF_MEMORY_QSPI_NOR_CODE_VERSION_MAJOR

#define SF_MEMORY_QSPI_NOR_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ SF_MEMORY_QSPI_NOR_Close()

ssp_err_t SF_MEMORY_QSPI_NOR_Close (sf_memory_ctrl_t *const *p_ctrl*)

Close the Memory QSPI NOR driver module.

Return values

SSP_SUCCESS	Close was successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- qspi_api_t::close

Validate the parameters

Close the underlying QSPI

Mark instance as closed for future reference

◆ SF_MEMORY_QSPI_NOR_Erase()

ssp_err_t SF_MEMORY_QSPI_NOR_Erase (sf_memory_ctrl_t * *p_ctrl*, uint32_t const *memory_address*, uint32_t const *num_bytes*)

Erase a number of bytes from the flash.

Return values

SSP_SUCCESS	The command to erase the flash was executed successfully.
SSP_ERR_ASSERTION	p_ctrl or memory_address is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid num_bytes entered.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_TIMEOUT	Wait timed out.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls
- qspi_api_t::erase
- qspi_api_t::statusGet

Validate the parameters

Check whether instance is open or not

Check whether the device address is valid

Calculate maximum erase size

Check whether we have valid erase size

◆ SF_MEMORY_QSPI_NOR_Flush()

ssp_err_t SF_MEMORY_QSPI_NOR_Flush (sf_memory_ctrl_t *const *p_ctrl*)

Flush any pending data to the disk. This is not required for QSPI NOR Flash.

Return values

SSP_SUCCESS	NO error detected.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Validate the parameters

Check whether instance is open or not

◆ SF_MEMORY_QSPI_NOR_InfoGet()

ssp_err_t SF_MEMORY_QSPI_NOR_InfoGet (sf_memory_ctrl_t *const *p_ctrl*, sf_memory_info_t *const *p_info*)

Returns the information about the flash.

Return values

SSP_SUCCESS	Memory info structure updated successfully.
SSP_ERR_ASSERTION	p_ctrl or p_info is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Returns

InfoGet status.

Validate the parameters

Memory info is obtained while opening, use it

◆ SF_MEMORY_QSPI_NOR_Open()

ssp_err_t SF_MEMORY_QSPI_NOR_Open (sf_memory_ctrl_t *const *p_ctrl*, sf_memory_cfg_t const *const *p_cfg*)

Open the SF Memory QSPI Nor driver module.

Open the SF Memory QSPI Nor driver module for the purposes of reading and writing flash memory.

Return values

SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_ALREADY_OPEN	Driver is already open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- qspi_api_t::open
- qspi_api_t::infoGet
- qspi_api_t::close

Validate the parameters

Check whether the framework is in already open state

Update instance control with the lower level driver and configuration information.

Open the QSPI driver

Update the memory info

Close QSPI in case of failure

Mark instance as open for future reference

◆ SF_MEMORY_QSPI_NOR_Read()

ssp_err_t SF_MEMORY_QSPI_NOR_Read (sf_memory_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *memory_address*, uint32_t const *num_bytes*)

Read data from the flash.

Read specified number of bytes of data from a particular address on the QSPI flash device.

Return values

SSP_SUCCESS	The data read was successful.
SSP_ERR_ASSERTION	p_ctrl,p_dest_address or memory_address is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_INVALID_ARGUMENT	Number of bytes requested are invalid.
SSP_ERR_TIMEOUT	Wait timed out.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- qspi_api_t::read
- qspi_api_t::statusGet

Validate the parameters

Check whether instance is open or not

Check whether the device address is valid

Read data from banks using ROM area.

◆ SF_MEMORY_QSPI_NOR_VersionGet()

ssp_err_t SF_MEMORY_QSPI_NOR_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

Returns

API and Code version.

Validate the parameters

◆ SF_MEMORY_QSPI_NOR_Write()

ssp_err_t SF_MEMORY_QSPI_NOR_Write (sf_memory_ctrl_t *const *p_ctrl*, uint8_t *const *p_src_address*, uint32_t const *memory_address*, uint32_t const *num_bytes*)

Program data to the flash.

Return values

SSP_SUCCESS	The flash was programmed successfully.
SSP_ERR_ASSERTION	p_ctrl,p_src_address or memory_address is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_TIMEOUT	Wait timed out.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- qspi_api_t::pageProgram
- qspi_api_t::statusGet

Validate the parameters

Check whether instance is open or not

Check whether the device address is valid

Check whether page size of QSPI NOR is larger than the read buffer size

Calculate current write size

Copy data from source address to a buffer for write if the source address is within address range of QSPI NOR

Program using underlying QSPI driver

Wait if there is any write operation in progress

sf_memory_qspi_nor_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Memory framework

#include <sf_memory_qspi_nor.h>

Data Fields

qspi_instance_t *	p_qspi QSPI instance.

qspi_info_t	qspi_info QSPI info.
uint32_t	open Track opened/closed status.
sf_memory_region_info_t	region_info Memory region info.
uint32_t	timeout_ticks Number of ticks to timeout on erase or write waiting.
sf_memory_qspi_nor_pending_operation_t	pending_operation The last operation sent to the QSPI chip.
uint32_t	pending_operation_size The size of the last operation.
void *	p_delay_callback_context Context passed to the delay callback.
void(*	p_delay_callback)(sf_memory_qspi_nor_delay_callback_args_t *p_args) Pointer to user callback function.

Detailed Description

Control block. DO NOT INITIALIZE. Initialization occurs when sf_memory_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_memory_qspi_nor.h

sf_memory_qspi_nor_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Memory framework

#include <sf_memory_qspi_nor.h>

Data Fields

qspi_instance_t const *const	p_qspi Lower level driver.
uint32_t	timeout_ticks Number of ticks to timeout on erase or write waiting.
void *	p_delay_callback_context Context passed to the delay callback.
void(*	p_delay_callback)(sf_memory_qspi_nor_delay_callback_args_t *p_args)

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ p_delay_callback

void(* sf_memory_qspi_nor_cfg_t::p_delay_callback) (sf_memory_qspi_nor_delay_callback_args_t *p_args)
A custom delay callback can be used to fine tune the amount of time to wait before starting to poll the QSPI chip after a write or erase operation. If a custom delay callback is used and a timeout occurs the p_args->timeout_remaining variable must be set to 0 to indicate to the calling function that a timeout has occurred.

The documentation for this struct was generated from the following file:

- sf_memory_qspi_nor.h

5.1.3.29 Messaging Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Messaging Framework implementation. More...

Data Structures

struct	sf_message_instance_ctrl_t

Macros

#define	SF_MESSAGE_CODE_VERSION_MAJOR (2U)
#define	SF_MESSAGE_QUEUE_MESSAGE_WORDS (1)

Functions

ssp_err_t	SF_MESSAGE_Open (sf_message_ctrl_t *const p_api_ctrl, sf_message_cfg_t const *const p_cfg) Initialize message framework. This function initiates the messaging framework control block, configures the work memory corresponding to the configuration parameters. More...
ssp_err_t	SF_MESSAGE_Close (sf_message_ctrl_t *const p_api_ctrl) Closes message framework. More...
ssp_err_t	SF_MESSAGE_BufferAcquire (sf_message_ctrl_t const *const p_api_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option) Acquire buffer for message passing from the block. More...
ssp_err_t	SF_MESSAGE_BufferRelease (sf_message_ctrl_t *const p_api_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option) Release buffer obtained by SF_MESSAGE_BufferAcquire(). More...
ssp_err_t	SF_MESSAGE_Post (sf_message_ctrl_t *const p_api_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option) Post a message to the subscribers. More...

ssp_err_t	SF_MESSAGE_Pend (sf_message_ctrl_t const *const p_api_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option) Pend on a message. More...
ssp_err_t	SF_MESSAGE_VersionGet (ssp_version_t *const p_version) Get the version of the messaging framework. Stores version information in provided pointer. More...

Detailed Description

RTOS-integrated Messaging Framework implementation.

Macro Definition Documentation

◆ SF_MESSAGE_CODE_VERSION_MAJOR

#define SF_MESSAGE_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ SF_MESSAGE_QUEUE_MESSAGE_WORDS

#define SF_MESSAGE_QUEUE_MESSAGE_WORDS (1)
The size of a message queue in words

Function Documentation

◆ SF_MESSAGE_BufferAcquire()

ssp_err_t SF_MESSAGE_BufferAcquire (sf_message_ctrl_t const *const *p_api_ctrl*, sf_message_header_t ** *pp_buffer*, sf_message_acquire_cfg_t const *const *p_acquire_cfg*, uint32_t const *wait_option*)

Acquire buffer for message passing from the block.

Return values

SSP_SUCCESS	Buffer acquisition was successful.
SSP_ERR_ASSERTION	p_ctrl, p_acquire_cfg or pp_buffer is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_INTERNAL	OS service call fails.

Note

This API function allows to be called from not only thread but also ISR.

Allocates buffer in the block memory pool.

Clears buffer control block

Sets the buffer in-use flag

Sets the address of the allocated buffer to 'pp_buffer'

Clears the event class and event code in the buffer. This is because the initial value in the buffer control block is unknown and it would not be safe.

Sets the 'buffer_keep' flag in the buffer control block if SF_MESSAGE_ACQUIRE_OPTION_KEEP is set to the 'option' argument

◆ SF_MESSAGE_BufferRelease()

ssp_err_t SF_MESSAGE_BufferRelease (sf_message_ctrl_t *const *p_api_ctrl*, sf_message_header_t *const *p_buffer*, sf_message_release_option_t const *option*)

Release buffer obtained by SF_MESSAGE_BufferAcquire().

Return values

SSP_SUCCESS	Buffer release was successful.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_ASSERTION	p_ctrl or p_buffer is NULL.
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	If buffer address is not aligned or p_buffer is not in the block pool range.
SSP_ERR_BUFFER_RELEASED	Buffer has been released.
SSP_ERR_INTERNAL	OS service call fails

Note

This API function allows to be called from thread but also from ISR.

Calculates the address of the buffer control block

Release buffer in the condition below. (1) The counting semaphore is zero and the buffer keep option is not specified. (2) 'option' is set to SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE.

Clears the flags in the buffer control block.

Release the buffer using ThreadX API "tx_block_release"

Set back the backed up interrupt mask level.

Invokes an user callback function if it is registered in the condition below. (1) The counting semaphore is zero. (2) 'option' is set to SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE.

Sets SF_MESSAGE_CALLBACK_EVENT_NAK if any subscribers for the message have responded NAK

Sets SF_MESSAGE_CALLBACK_EVENT_ACK if all subscribers for the message have responded ACK

Sets the pointer to the context to kept in the buffer control block

Invokes the registered user callback function.

◆ SF_MESSAGE_Close()

ssp_err_t SF_MESSAGE_Close (sf_message_ctrl_t *const *p_api_ctrl*)

Closes message framework.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.

Note

This API function only allows to be called from thread context.

Finds subscribers and flushes their queues

Deletes memory pools allocated in the work memory

◆ SF_MESSAGE_Open()

ssp_err_t SF_MESSAGE_Open (sf_message_ctrl_t *const *p_api_ctrl*, sf_message_cfg_t const *const *p_cfg*)

Initialize message framework. This function initiates the messaging framework control block, configures the work memory corresponding to the configuration parameters.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	p_ctrl, p_cfg or p_cfg->p_work_memory_start is NULL.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_IN_USE	The Messaging Framework is in use.
SSP_ERR_INVALID_WORKBUFFER_SIZE	Invalid work buffer size.
SSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.

Note

This API function is allowed to be called once per instance. The behavior if called twice is undefined.
This API function only allows to be called from thread context.

Creates the memory pools in the work memory area

Registers subscriber lists

Changes the messaging framework status from CLOSED to OPENED

◆ SF_MESSAGE_Pend()

ssp_err_t SF_MESSAGE_Pend (sf_message_ctrl_t const *const *p_api_ctrl*, TX_QUEUE const *const *p_queue*, sf_message_header_t ** *pp_buffer*, uint32_t const *wait_option*)

Pend on a message.

Return values

SSP_SUCCESS	Message pending was successful.
SSP_ERR_ASSERTION	p_ctrl, pp_buffer or p_queue is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_MESSAGE_QUEUE_EMPTY	Queue is empty.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_INTERNAL	OS service call fails.

Note

This API function allows to be called from not only thread but also ISR(if wait_option is TX_NO_WAIT).

Receiving message here. Receiving data is not message itself but the pointer to the buffer

If there is no data in the message queue and TX_NO_WAIT is specified to wait_option, return immediately with SSP_ERR_MESSAGE_QUEUE_EMPTY error code

◆ SF_MESSAGE_Post()

ssp_err_t SF_MESSAGE_Post (sf_message_ctrl_t *const *p_api_ctrl*, sf_message_header_t const *const *p_buffer*, sf_message_post_cfg_t const *const *p_post_cfg*, sf_message_post_err_t *const *p_post_err*, uint32_t const *wait_option*)

Post a message to the subscribers.

Return values

SSP_SUCCESS	Message posting was successful.
SSP_ERR_ASSERTION	p_ctrl or p_buffer is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_NO_SUBSCRIBER_FOUND	No subscriber found.
SSP_ERR_BUFFER_RELEASED	Buffer has been released.
SSP_ERR_MESSAGE_QUEUE_FULL	Queue is full (Timeout occurs before sending a message if timeout option is specified)
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	If buffer address is not aligned or p_buffer is not in the block pool range.
SSP_ERR_INTERNAL	OS service call fails

Note

This API function allows to be called from not only thread but also ISR(if wait_option is TX_NO_WAIT). Another buffer writing to the buffer before the message read by message consumers results data overwriting.

- Checks the number of the subscribers of specified event class

Calculates the address of the buffer control block

Counts up the counting semaphore in the buffer control block

Registers user callback function and context passed from user

◆ SF_MESSAGE_VersionGet()

ssp_err_t SF_MESSAGE_VersionGet (ssp_version_t *const *p_version*)

Get the version of the messaging framework. Stores version information in provided pointer.

Return values

SSP_SUCCESS	Got version number successfully.
SSP_ERR_ASSERTION	p_version is NULL.

sf_message_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Messaging Framework

#include <sf_message.h>

Data Fields

TX_BLOCK_POOL	block_pool Pointer to the memory block pool control.
sf_message_subscriber_list_t **	pp_subscriber_lists Pointer array to the subscriber lists.
uint32_t	buffer_size Bytes of the message buffer.
uint32_t	number_of_buffers The number of allocated buffers.
uint16_t	number_of_subscriber_groups The number of subscriber groups.
sf_message_state_t	state Status of the message framework.

Detailed Description

Messaging framework instance control block structure

The documentation for this struct was generated from the following file:

- sf_message.h

5.1.3.30 Power Profiles Framework V2

Renesas Synergy Software Package Reference » Framework Layer

Power Profiles Framework. More...

Macros

#define	SF_POWER_PROFILES_V2_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	SF_POWER_PROFILES_V2_Open (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_cfg_t const *const p_cfg) Configures the Power Profiles framework and opens any required HAL layer drivers that will be used. More...
ssp_err_t	SF_POWER_PROFILES_V2_RunApply (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_run_cfg_t const *const p_cfg) Applies a Run profile. More...
ssp_err_t	SF_POWER_PROFILES_V2_LowPowerApply (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_low_power_cfg_t const *const p_cfg) Applies a Low Power profile. More...
ssp_err_t	SF_POWER_PROFILES_V2_Close (sf_power_profiles_v2_ctrl_t *const p_ctrl) Closes the framework. More...
ssp_err_t	SF_POWER_PROFILES_V2_VersionGet (ssp_version_t *const p_version) Gets version and stores it in provided pointer p_version. More...

Detailed Description

Power Profiles Framework.

Macro Definition Documentation

◆ SF_POWER_PROFILES_V2_CODE_VERSION_MAJOR

#define SF_POWER_PROFILES_V2_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ SF_POWER_PROFILES_V2_Close()

ssp_err_t SF_POWER_PROFILES_V2_Close (sf_power_profiles_v2_ctrl_t *const *p_ctrl*)

Closes the framework.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.

Clear information from control block so other functions know this block is closed.

◆ SF_POWER_PROFILES_V2_LowPowerApply()

ssp_err_t SF_POWER_PROFILES_V2_LowPowerApply (sf_power_profiles_v2_ctrl_t *const *p_ctrl*, sf_power_profiles_v2_low_power_cfg_t const *const *p_cfg*)

Applies a Low Power profile.

The SF_POWER_PROFILES_V2_LowPowerApply function will:

- Apply a LPMv2 configuration to prepare for low power
- Apply an IO port configuration to prepare for low power, if supplied
- Notify application that low power is about to be entered
- Enter low power mode
- When low power mode is exited, apply an IO port configuration for wake up, if supplied
- Notify application that wake up has occurred

Return values

SSP_SUCCESS	Entered and exited low power mode successfully.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.
SSP_ERR_UNSUPPORTED	This function is not supported by one of the HAL drivers, r_lpmv2, r_ioport.
SSP_ERR_INVALID_MODE	r_lpmv2 mode is not LPMV2_LOW_POWER_MODE_SLEEP but r_lmv2 p_extend is NULL.
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.

Returns

See Common Error Codes, r_ioport, or r_lpmv2 drivers for other possible return codes or causes.

Apply the LPM configuration.

Apply the pre-low power IOPORT configuration.

Notify application.

Enter low power mode.

Apply the post-low power IOPORT configuration.

Notify application.

◆ SF_POWER_PROFILES_V2_Open()

ssp_err_t SF_POWER_PROFILES_V2_Open (sf_power_profiles_v2_ctrl_t *const *p_ctrl*, sf_power_profiles_v2_cfg_t const *const *p_cfg*)

Configures the Power Profiles framework and opens any required HAL layer drivers that will be used.

The SF_POWER_PROFILES_V2_Open function initializes the critical data structures and variables.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_IN_USE	Power profiles framework is already open.
SSP_ERR_INTERNAL	Unable to obtain mutex. Unable to release mutex.

Returns

See Common Error Codes for other possible return codes or causes.

Initialize the LPM HAL driver.

Mark control block open so other tasks know it is valid.

◆ SF_POWER_PROFILES_V2_RunApply()

ssp_err_t SF_POWER_PROFILES_V2_RunApply (sf_power_profiles_v2_ctrl_t *const *p_ctrl*, sf_power_profiles_v2_run_cfg_t const *const *p_cfg*)

Applies a Run profile.

The SF_POWER_PROFILES_V2_RunApply function will:

- Apply an IO port configuration, if supplied
- Apply a clock configuration

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_INVALID_ARGUMENT	Clock configuration is invalid.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.

Returns

See Common Error Codes, r_ioport, or r_cgc driver for other possible return codes or causes.

Apply the ioport configuration.

Set the clock config, this also sets the operating mode based on the clock speed.

◆ SF_POWER_PROFILES_V2_VersionGet()

ssp_err_t SF_POWER_PROFILES_V2_VersionGet (ssp_version_t *const *p_version*)

Gets version and stores it in provided pointer p_version.

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

5.1.3.31 SPI Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated SPI Framework. More...

Data Structures

struct	sf_spi_instance_ctrl_t

Functions

ssp_err_t	SF_SPI_Open (sf_spi_ctrl_t *const p_api_ctrl, sf_spi_cfg_t const *const p_cfg) Initialize a SPI bus and open low level SPI driver. More...
ssp_err_t	SF_SPI_Read (sf_spi_ctrl_t *const p_api_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Starts the transfer process and receives data from SPI device. More...
ssp_err_t	SF_SPI_Write (sf_spi_ctrl_t *const p_api_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Starts the transfer process and writes data to SPI device. More...
ssp_err_t	SF_SPI_WriteRead (sf_spi_ctrl_t *const p_api_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout) Simultaneously transmit data to SPI device while receiving data from SPI device(full duplex). More...
ssp_err_t	SF_SPI_Close (sf_spi_ctrl_t *const p_api_ctrl) Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. More...
ssp_err_t	SF_SPI_Lock (sf_spi_ctrl_t *const p_api_ctrl) Lock the bus for a device. More...
ssp_err_t	SF_SPI_Unlock (sf_spi_ctrl_t *const p_api_ctrl) Unlock the bus for a particular device and make the bus usable for other devices. More...
ssp_err_t	SF_SPI_VersionGet (ssp_version_t *const p_version)

	Get the version information of the framework. More...
ssp_err_t	SF_SPI_LockWait (sf_spi_ctrl_t *const p_api_ctrl, uint32_t const timeout) Lock the SPI bus resource. Once bus is locked by a device it can not be used by other devices. More...

Detailed Description

RTOS-integrated SPI Framework.

Function Documentation

◆ SF_SPI_Close()

ssp_err_t SF_SPI_Close (sf_spi_ctrl_t *const *p_api_ctrl*)

Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus.

Return values

SSP_SUCCESS	SPI channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device not opened.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- spi_api_t::close

Note

This function is reentrant for any device.

Check whether device is open.

Acquire the device count mutex before accessing the shared resource in close.

Check the count of opened devices on the bus. If there are no devices opened or all other devices on the bus are closed then close the low level SPI driver and release the RTOS services used by the bus.

Get the low level control in use.

Close low level driver.

Delete RTOS services used by the bus.

Decrement device count.

Delete the device count mutex

Decrement device count.

Release the device count mutex

Set device to closed state

◆ SF_SPI_Lock()

ssp_err_t SF_SPI_Lock (sf_spi_ctrl_t *const *p_api_ctrl*)

Lock the bus for a device.

Return values

SSP_SUCCESS	SPI bus is successfully locked.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.

Note

This function is reentrant for any device.

Check whether device is open.

Get the mutex for this device.

Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.

Release the mutex

Set lock flag.

◆ SF_SPI_LockWait()

ssp_err_t SF_SPI_LockWait (sf_spi_ctrl_t *const *p_api_ctrl*, uint32_t const *timeout*)

Lock the SPI bus resource. Once bus is locked by a device it can not be used by other devices.

Return values

SSP_SUCCESS	SPI channel is successfully locked within the specified timeout.
SSP_ERR_ASSERTION	Pointer to SPI control block is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_TIMEOUT	Mutex not available in timeout.

Note

This function is reentrant for any device.

Check whether device is open.

Get the mutex for this device.

Set lock flag.

Enable slave.

◆ SF_SPI_Open()

ssp_err_t SF_SPI_Open (sf_spi_ctrl_t *const *p_api_ctrl*, sf_spi_cfg_t const *const *p_cfg*)

Initialize a SPI bus and open low level SPI driver.

Return values

SSP_SUCCESS	SPI channel is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_cfg, Pointer to Open, Close, Read, Write, or Writeread API interfaces, p_cfg->p_bus or p_cfg->p_lower_lvl_cfg.
SSP_ERR_INTERNAL	Internal error occurred.
SSP_ERR_ALREADY_OPEN	Same SPI framework device is already open.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is

- spi_api_t::open

Note

This function is reentrant for any channel.
Control block must be cleared by caller before calling this function.

Check whether device is already opened or not.

Copy bus to control

Copy chip_select to control

Copy chip_select level to control

Initialize bus lock to false

Set framework level callback function.

Save context for use in ISRs.

Use bus channel in device open.

Enter a critical section before checking the device count mutex status.

Check if device count mutex is already created. If not then create the mutex.

Create device_count_mutex. This is used to protect shared variable device_count in bus control structure.

If mutex create fails, return error.

Exit critical section

Acquire the device count mutex before accessing the shared resource. Try again if the mutex was deleted in close.

Increment device count.

Save device configuration for reconfiguration.

Set device state as Opened.

Initialize chip select.

◆ SF_SPI_Read()

ssp_err_t SF_SPI_Read (sf_spi_ctrl_t *const *p_api_ctrl*, void *const *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*, uint32_t const *timeout*)

Starts the transfer process and receives data from SPI device.

Return values

SSP_SUCCESS	Data read completed successfully.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_dest, length.
SSP_ERR_NOT_OPEN	Device not opened.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- spi_api_t::read

Check whether device is open.

Get mutex for this bus.

Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.

Release the mutex

Perform read.

Finish transfer.

◆ SF_SPI_Unlock()

ssp_err_t SF_SPI_Unlock (sf_spi_ctrl_t *const *p_api_ctrl*)

Unlock the bus for a particular device and make the bus usable for other devices.

Return values

SSP_SUCCESS	SPI bus is successfully unlocked.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.

Note

This function is reentrant for any device.

Check whether device is open.

Acquire the mutex.

Clear lock flag.

Disable slave.

Release the mutex so that others can use the bus.

◆ SF_SPI_VersionGet()

ssp_err_t SF_SPI_VersionGet (ssp_version_t *const *p_version*)

Get the version information of the framework.

Return values

SSP_ERR_ASSERTION	p_version is NULL.
SSP_SUCCESS	Successful return.

Checks error. Further parameter checking can be done at the driver layer.

◆ SF_SPI_Write()

ssp_err_t SF_SPI_Write (sf_spi_ctrl_t *const *p_api_ctrl*, void *const *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*, uint32_t const *timeout*)

Starts the transfer process and writes data to SPI device.

Return values

SSP_SUCCESS	Data write completed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl, p_src, length.
SSP_ERR_NOT_OPEN	Device not opened.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- spi_api_t::write

Check whether device is open.

Get mutex for this bus.

Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.

Release the mutex

Perform write.

Finish transfer.

◆ SF_SPI_WriteRead()

ssp_err_t SF_SPI_WriteRead (sf_spi_ctrl_t *const *p_api_ctrl*, void *const *p_src*, void *const *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*, uint32_t const *timeout*)

Simultaneously transmit data to SPI device while receiving data from SPI device(full duplex).

Return values

SSP_SUCCESS	Data write completed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl, p_src, p_dest, length.
SSP_ERR_NOT_OPEN	Device not opened.

Returns

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- spi_api_t::writeRead

Check whether device is open.

Get mutex for this bus.

Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.

Release the mutex

Perform write read.

Finish transfer.

sf_spi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » SPI Framework

#include <sf_spi.h>

Data Fields

sf_spi_bus_t *	p_bus Bus using this device (copy from cfg)
ioport_port_pin_t	chip_select Chip select for this device (copy from cfg)
ioport_level_t	chip_select_level_active

	Polarity of CS, active High or Low (copy from cfg)
spi_cfg_t	lower_lvl_cfg SPI peripheral configuration, use for bus reconfiguration.
spi_ctrl_t *	p_lower_lvl_ctrl SPI peripheral control block.
sf_spi_dev_state_t	dev_state Device status.
bool	locked Lock and unlock bus for a device.

Detailed Description

SPI device context. DO NOT INITIALIZE. Initialization occurs when sf_spi_api_t::open is called.

The documentation for this struct was generated from the following file:

- sf_spi.h

5.1.3.32 Thread Monitor Framework

Renesas Synergy Software Package Reference » Framework Layer

Framework module providing monitoring of threads. More...

Data Structures

struct	sf_thread_monitor_instance_ctrl_t

Macros

#define	SF_THREAD_MONITOR_CODE_VERSION_MAJOR (2U)

Functions

void	SF_THREAD_MONITOR_Thread (ULONG thread_input)

	The SF_THREAD_MONITOR_thread() is the main thread monitor thread which runs periodically. More...
ssp_err_t	SF_THREAD_MONITOR_Open (sf_thread_monitor_ctrl_t *const p_api_ctrl, sf_thread_monitor_cfg_t const *const p_cfg) Calls the driver .open function in the p_lower_lvl_wdt parameter. More...
ssp_err_t	SF_THREAD_MONITOR_Close (sf_thread_monitor_ctrl_t *const p_api_ctrl) Stop the thread monitoring thread. All threads are unregistered and no longer monitored. More...
ssp_err_t	SF_THREAD_MONITOR_ThreadRegister (sf_thread_monitor_ctrl_t *const p_api_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max) Register a thread to be monitored by the watchdog monitoring thread. More...
ssp_err_t	SF_THREAD_MONITOR_ThreadUnregister (sf_thread_monitor_ctrl_t *const p_api_ctrl) Remove a thread from being monitored by the Watchdog monitoring thread. More...
ssp_err_t	SF_THREAD_MONITOR_CountIncrement (sf_thread_monitor_ctrl_t *const p_api_ctrl) Safely increment the counter of a thread. A mutex is used to ensure this increment occurs without corruption. More...
ssp_err_t	SF_THREAD_MONITOR_VersionGet (ssp_version_t *const p_version) Get version and store it in provided pointer p_version. Implements sf_thread_monitor_api_t::versionGet. More...

Detailed Description

Framework module providing monitoring of threads.

Any misbehaving threads result in the device being reset. Both the WDT and IWDT HAL modules are supported by this framework module.

Interface Used

See also
WDT Interface

Macro Definition Documentation

◆ SF_THREAD_MONITOR_CODE_VERSION_MAJOR

#define SF_THREAD_MONITOR_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Function Documentation

◆ SF_THREAD_MONITOR_Close()

ssp_err_t SF_THREAD_MONITOR_Close (sf_thread_monitor_ctrl_t *const *p_api_ctrl*)

Stop the thread monitoring thread. All threads are unregistered and no longer monitored.

Return values

SSP_SUCCESS	Close was successful and watchdog monitoring thread has stopped.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open() has either not been called or it was not called successfully.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Note

This function is not reentrant.
This function DOES NOT stop the watchdog timer (e.g. WDT or IWDT). It does however stop the thread which refreshes the these timers. To prevent the hardware watchdog from reseting the device the watchdog must be refreshed elsewhere.
The Thread Monitor can be closed with SF_THREAD_MONITOR_Close(). However, if the underlying watchdog timer hardware such as WDT or IWDT cannot be closed or stopped then there are likely to be undesirable results.

◆ SF_THREAD_MONITOR_CountIncrement()

ssp_err_t SF_THREAD_MONITOR_CountIncrement (sf_thread_monitor_ctrl_t *const *p_api_ctrl*)

Safely increment the counter of a thread. A mutex is used to ensure this increment occurs without corruption.

Return values

SSP_SUCCESS	Thread counter successfully incremented.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads to be monitored array to add this thread.
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open() has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Note

This function is reentrant.

◆ SF_THREAD_MONITOR_Open()

ssp_err_t SF_THREAD_MONITOR_Open (sf_thread_monitor_ctrl_t *const *p_api_ctrl*, sf_thread_monitor_cfg_t const *const *p_cfg*)

Calls the driver .open function in the p_lower_lvl_wdt parameter.

After successively opening and configuring the HAL driver, the SF_THREAD_MONITOR_Open() function starts the thread monitoring thread. This thread is responsible for checking thread execution through thread count variables and for refreshing the implemented watchdog timer peripheral.

Return values

SSP_SUCCESS	Initialization was successful and watchdog monitoring thread has started.
SSP_ERR_ASSERTION	p_ctrl, p_cfg, p_cfg->p_lower_lvl_wdt pointers and/or p_cfg->p_lower_lvl_wdt are NULL.
SSP_ERR_IN_USE	Thread monitor has already been opened.
SSP_ERR_INVALID_MODE	Low level watchdog peripheral returned an error when opened.
SSP_ERR_UNSUPPORTED	Data structure could not be allocated.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid for the low level driver.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Note

This function is not reentrant.
If there are no threads registered to be monitored the monitoring thread refreshes the watchdog and prevents the watchdog from reseting the device.
The Thread Monitor can be closed with SF_THREAD_MONITOR_Close and then SF_THREAD_MONITOR_Open can be called again. However, if the underlying watchdog timer hardware such as WDT or IWDT cannot be closed or stopped then there are likely to be undesirable results.

◆ SF_THREAD_MONITOR_Thread()

void SF_THREAD_MONITOR_Thread (ULONG *thread_input*)

The SF_THREAD_MONITOR_thread() is the main thread monitor thread which runs periodically.

The period is determined from the timeout period of the Watchdog hardware. This thread's period should be half of that of the watchdog hardware so the watchdog is refreshed at 50% of the count value.

Parameters

[in]	thread_input	This is the address of the control block for the thread monitor module. This allows this thread to be created multiple times with the data used and hardware interfaced with governed by the control structure.

As this is a ThreadX thread this function does not return.

Note

This function is not reentrant.
If there are no threads registered to be monitored the monitoring thread will refresh/tickle/kick the watchdog and so prevent the watchdog from reseting the device.

◆ SF_THREAD_MONITOR_ThreadRegister()

ssp_err_t SF_THREAD_MONITOR_ThreadRegister (sf_thread_monitor_ctrl_t *const *p_api_ctrl*, sf_thread_monitor_counter_min_max_t const * *p_counter_min_max*)

Register a thread to be monitored by the watchdog monitoring thread.

This thread must supply the minimum and maximum expected values for the thread. The mininimum and maximum values can be determined by using monitoring mode.

Return values

SSP_SUCCESS	Thread successfully registered.
SSP_ERR_ASSERTION	p_ctrl or p_counter_min_max pointers are NULL.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads to be monitored array to add this thread. Increase the value of THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS in sf_thread_moinitor_cfg.h
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open() has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Note

This function is reentrant.

◆ SF_THREAD_MONITOR_ThreadUnregister()

ssp_err_t SF_THREAD_MONITOR_ThreadUnregister (sf_thread_monitor_ctrl_t *const *p_api_ctrl*)

Remove a thread from being monitored by the Watchdog monitoring thread.

Return values

SSP_SUCCESS	Thread successfully unregistered.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open() has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes.

Note

This function is reentrant.

◆ SF_THREAD_MONITOR_VersionGet()

ssp_err_t SF_THREAD_MONITOR_VersionGet (ssp_version_t *const *p_version*)
Get version and store it in provided pointer p_version. Implements sf_thread_monitor_api_t::versionGet. **Return values** SSP_SUCCESS — Version returned successfully. SSP_ERR_ASSERTION — Parameter p_version was null.

sf_thread_monitor_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Thread Monitor Framework

#include <sf_thread_monitor.h>

Public Member Functions

uint8_t stack [THREAD_MONITOR_THREAD_STACK_SIZE]	BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)

Data Fields

uint32_t	open
wdt_instance_t const *	p_lower_lvl_wdt
uint32_t	timeout_period_msec
uint32_t	timeout_period_watchdog_clocks
bool	profiling_mode_enabled
TX_MUTEX	mutex Mutex to protect access to the thread counters.
uint32_t	profiling_mode_check
sf_thread_monitor_thread_counter_t	thread_counters [THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS]
TX_THREAD	thread

	Thread monitor thread.
void const *	p_extend Extended configuration data.

Detailed Description

Thread monitor control block. DO NOT INITIALIZE. Initialization occurs when sf_thread_monitor_api_t::open is called.

Member Function Documentation

◆ BSP_ALIGN_VARIABLE_V2()

uint8_t stack [THREAD_MONITOR_THREAD_STACK_SIZE] sf_thread_monitor_instance_ctrl_t::BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)
Stack for thread monitor thread.

Field Documentation

◆ open

uint32_t sf_thread_monitor_instance_ctrl_t::open
Used by driver to check if the control structure is valid

◆ p_lower_lvl_wdt

wdt_instance_t const* sf_thread_monitor_instance_ctrl_t::p_lower_lvl_wdt
Pointer to interface structure for the watchdog peripheral

◆ profiling_mode_check

uint32_t sf_thread_monitor_instance_ctrl_t::profiling_mode_check
Value used to verify profiling mode is enabled when prfiling_mode_enabled == true.

◆ profiling_mode_enabled

bool sf_thread_monitor_instance_ctrl_t::profiling_mode_enabled
Used by the driver to check if profiling mode is enabled.

◆ thread_counters

sf_thread_monitor_thread_counter_t sf_thread_monitor_instance_ctrl_t::thread_counters[THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS]
Data storage for the thread counter information.

◆ timeout_period_msec

uint32_t sf_thread_monitor_instance_ctrl_t::timeout_period_msec
Time in milliseconds of the watchdog timeout period. Used to calculate the period of the monitoring thread.

◆ timeout_period_watchdog_clocks

uint32_t sf_thread_monitor_instance_ctrl_t::timeout_period_watchdog_clocks
Maximum count value of the watchdog. Used to synchronise to the count.

The documentation for this struct was generated from the following file:

- sf_thread_monitor.h

5.1.3.33 CTSU V2 Framework

Renesas Synergy Software Package Reference » Framework Layer

CTSU V2 Framework. More...

Data Structures

struct	sf_touch_ctsu_button_info_t
struct	sf_touch_ctsu_slider_info_t
struct	sf_touch_ctsu_wheel_info_t
struct	sf_touch_ctsu_instance_ctrl_t

Functions

ssp_err_t	SF_TOUCH_CTSU_Open (sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)

	Opens and configures the TOUCH Middle module. Implements sf_touch_ctsu_api_t::open. More...
ssp_err_t	SF_TOUCH_CTSU_ScanStart (sf_touch_ctsu_ctrl_t *const p_ctrl) This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with SF_TOUCH_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements sf_touch_ctsu_api_t::scanStart. More...
ssp_err_t	SF_TOUCH_CTSU_DataGet (sf_touch_ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position) Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements sf_touch_ctsu_api_t::dataGet. More...
ssp_err_t	SF_TOUCH_CTSU_CallbackSet (sf_touch_ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(sf_touch_ctsu_callback_args_t *), void const *const p_context, sf_touch_ctsu_callback_args_t *const p_callback_memory)
ssp_err_t	SF_TOUCH_CTSU_Close (sf_touch_ctsu_ctrl_t *const p_ctrl) Disables specified TOUCH control block. Implements sf_touch_ctsu_api_t::close. More...
ssp_err_t	SF_TOUCH_CTSU_VersionGet (ssp_version_t *const p_version)

Detailed Description

CTSU V2 Framework.

Function Documentation

◆ SF_TOUCH_CTSU_CallbackSet()

ssp_err_t SF_TOUCH_CTSU_CallbackSet (sf_touch_ctsu_ctrl_t *const *p_api_ctrl*, void(*)(sf_touch_ctsu_callback_args_t *) *p_callback*, void const *const *p_context*, sf_touch_ctsu_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements sf_touch_ctsu_api_t::callbackSet

Return values

SSP_SUCCESS	Callback updated successfully.
SSP_ERR_ASSERTION	A required pointer is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened.

◆ SF_TOUCH_CTSU_Close()

ssp_err_t SF_TOUCH_CTSU_Close (sf_touch_ctsu_ctrl_t *const *p_ctrl*)

Disables specified TOUCH control block. Implements sf_touch_ctsu_api_t::close.

Return values

SSP_SUCCESS	Successfully closed.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Open to configure.

◆ SF_TOUCH_CTSU_DataGet()

ssp_err_t SF_TOUCH_CTSU_DataGet (sf_touch_ctsu_ctrl_t *const *p_ctrl*, uint64_t * *p_button_status*, uint16_t * *p_slider_position*, uint16_t * *p_wheel_position*)

Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements sf_touch_ctsu_api_t::dataGet.

Return values

SSP_SUCCESS	Successfully data decoded.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.
SSP_ERR_CTSU_SCANNING	Scanning this instance.
SSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.

◆ SF_TOUCH_CTSU_Open()

ssp_err_t SF_TOUCH_CTSU_Open (sf_touch_ctsu_ctrl_t *const *p_ctrl*, sf_touch_ctsu_cfg_t const *const *p_cfg*)

Opens and configures the TOUCH Middle module. Implements sf_touch_ctsu_api_t::open.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
SSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Parameter Setting

◆ SF_TOUCH_CTSU_ScanStart()

ssp_err_t SF_TOUCH_CTSU_ScanStart (sf_touch_ctsu_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with SF_TOUCH_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements sf_touch_ctsu_api_t::scanStart.

Return values

SSP_SUCCESS	Successfully started.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.
SSP_ERR_CTSU_SCANNING	Scanning this instance or other.
SSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.

◆ SF_TOUCH_CTSU_VersionGet()

ssp_err_t SF_TOUCH_CTSU_VersionGet (ssp_version_t *const *p_version*)
Return TOUCH Middle module version. Implements sf_touch_ctsu_api_t::versionGet. **Return values** SSP_SUCCESS — Version returned successfully. SSP_ERR_ASSERTION — Null pointer passed as a parameter

sf_touch_ctsu_button_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » CTSU V2 Framework

#include <sf_touch_ctsuv2.h>

Data Fields

uint64_t	status Touch result bitmap.
uint16_t *	p_threshold Pointer to Threshold value array. g_touch_button_threshold[] is set by Open API.
uint16_t *	p_hysteresis Pointer to Hysteresis value array. g_touch_button_hysteresis[] is set by Open API.
uint16_t *	p_reference Pointer to Reference value array. g_touch_button_reference[] is set by Open API.
uint16_t *	p_on_count Continuous touch counter. g_touch_button_on_count[] is set by Open API.
uint16_t *	p_off_count

	Continuous non-touch counter. g_touch_button_off_count[] is set by Open API.
uint32_t *	p_drift_buf Drift reference value. g_touch_button_drift_buf[] is set by Open API.
uint16_t *	p_drift_count Drift counter. g_touch_button_drift_count[] is set by Open API.
uint8_t	on_freq Copy from config by Open API.
uint8_t	off_freq Copy from config by Open API.
uint16_t	drift_freq Copy from config by Open API.
uint16_t	cancel_freq Copy from config by Open API.

Detailed Description

Information of button

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2.h

sf_touch_ctsu_slider_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » CTSU V2 Framework

#include <sf_touch_ctsuv2.h>

Data Fields

uint16_t *	p_position Calculated Position data. g_touch_slider_position[] is set by Open API.
uint16_t *	p_threshold Copy from config by Open API. g_touch_slider_threshold[] is set by Open API.

Detailed Description

Information of slider

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2.h

sf_touch_ctsu_wheel_info_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » CTSU V2 Framework

#include <sf_touch_ctsuv2.h>

Data Fields

uint16_t *	p_position Calculated Position data. g_touch_wheel_position[] is set by Open API.
uint16_t *	p_threshold Copy from config by Open API. g_touch_wheel_threshold[] is set by Open API.

Detailed Description

Information of wheel

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2.h

sf_touch_ctsu_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » CTSU V2 Framework

#include <sf_touch_ctsuv2.h>

Data Fields

uint32_t	open Whether or not driver is open.
sf_touch_ctsu_button_info_t	binfo Information of button.
sf_touch_ctsu_slider_info_t	sinfo Information of slider.
sf_touch_ctsu_wheel_info_t	winfo Information of wheel.
sf_touch_ctsu_cfg_t const *	p_touch_cfg Pointer to initial configurations.
ctsu_instance_t const *	p_ctsu_instance Pointer to CTSU instance.

Detailed Description

SF_TOUCH_CTSU private control block. DO NOT MODIFY. Initialization occurs when SF_TOUCH_CTSU_Open() is called.

The documentation for this struct was generated from the following file:

- sf_touch_ctsuv2.h

5.1.3.34 Touch Panel V2 Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated touch panel V2 Framework implementation touch chips. More...

Data Structures

struct	sf_touch_panel_v2_instance_ctrl_t

Macros

#define	SF_TOUCH_PANEL_V2_STACK_SIZE (SF_TOUCH_PANEL_V2_CFG_THREAD_STACK_SIZE)
#define	SF_TOUCH_PANEL_V2_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_version)
#define	SF_TOUCH_PANEL_V2_OPEN (0x54504e4cU)

Functions

ssp_err_t	SF_TOUCH_PANEL_V2_Open (sf_touch_panel_v2_ctrl_t *const p_api_ctrl, sf_touch_panel_v2_cfg_t const *const p_cfg) Implements sf_touch_panel_v2_api_t::open. More...
ssp_err_t	SF_TOUCH_PANEL_V2_Calibrate (sf_touch_panel_v2_ctrl_t *const p_api_ctrl, sf_touch_panel_v2_calibrate_t const *const p_display, sf_touch_panel_v2_calibrate_t const *const p_touchscreen, ULONG const timeout) Implements sf_touch_panel_v2_api_t::calibrate. More...
ssp_err_t	SF_TOUCH_PANEL_V2_Start (sf_touch_panel_v2_ctrl_t *const p_api_ctrl) Implements sf_touch_panel_v2_api_t::start. More...
ssp_err_t	SF_TOUCH_PANEL_V2_TouchDataGet (sf_touch_panel_v2_ctrl_t *const p_api_ctrl, sf_touch_panel_v2_payload_t *p_payload, ULONG const timeout) Implements sf_touch_panel_v2_api_t::touchDataGet. More...
ssp_err_t	SF_TOUCH_PANEL_V2_Stop (sf_touch_panel_v2_ctrl_t *const

	p_api_ctrl) Implements sf_touch_panel_v2_api_t::stop. More...
ssp_err_t	SF_TOUCH_PANEL_V2_Reset (sf_touch_panel_v2_ctrl_t *const p_api_ctrl) Implements sf_touch_panel_v2_api_t::reset. More...
ssp_err_t	SF_TOUCH_PANEL_V2_Close (sf_touch_panel_v2_ctrl_t *const p_api_ctrl) Implements sf_touch_panel_v2_api_t::close. More...
ssp_err_t	SF_TOUCH_PANEL_V2_VersionGet (ssp_version_t *const p_version) Implements sf_touch_panel_v2_api_t::versionGet. More...

Detailed Description

RTOS-integrated touch panel V2 Framework implementation touch chips.

Summary

This is a ThreadX touch panel framework implemented for external touch controllers with IRQ pins used to notify the application when new data is available.

Macro Definition Documentation

◆ SF_TOUCH_PANEL_V2_ERROR_RETURN

#define SF_TOUCH_PANEL_V2_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_version)
Macro for error logger.

◆ SF_TOUCH_PANEL_V2_OPEN

#define SF_TOUCH_PANEL_V2_OPEN (0x54504e4cU)
"TPNL" in ASCII, used to identify touch panel handle

◆ SF_TOUCH_PANEL_V2_STACK_SIZE

#define SF_TOUCH_PANEL_V2_STACK_SIZE (SF_TOUCH_PANEL_V2_CFG_THREAD_STACK_SIZE)
Stack size for touch panel thread.

Function Documentation

◆ SF_TOUCH_PANEL_V2_Calibrate()

ssp_err_t SF_TOUCH_PANEL_V2_Calibrate (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*, sf_touch_panel_v2_calibrate_t const *const *p_display*, sf_touch_panel_v2_calibrate_t const *const *p_touchscreen*, ULONG const *timeout*)

Implements sf_touch_panel_v2_api_t::calibrate.

Return values

SSP_SUCCESS	Touch panel calibrated successfully.
SSP_ERR_ASSERTION	A pointer parameter was NULL.
SSP_ERR_CALIBRATE_FAILED	Failed to calibrate
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.
SSP_ERR_INVALID_ARGUMENT	Set of display or touch screen coordinates passed are invalid.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Note

This function is reentrant for any panel.

Timeout not used in this implementation.

◆ SF_TOUCH_PANEL_V2_Close()

ssp_err_t SF_TOUCH_PANEL_V2_Close (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*)

Implements sf_touch_panel_v2_api_t::close.

Return values

SSP_SUCCESS	Touch panel instance successfully closed.
SSP_ERR_ASSERTION	Parameter p_api_ctrl was NULL, or a lower level driver reported this error.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.

Note

This function is reentrant.

Close the lower level driver.

Suspend internal thread.

Delete RTOS services used

Mark control block close

◆ SF_TOUCH_PANEL_V2_Open()

ssp_err_t SF_TOUCH_PANEL_V2_Open (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*, sf_touch_panel_v2_cfg_t const *const *p_cfg*)

Implements sf_touch_panel_v2_api_t::open.

Return values

SSP_SUCCESS	Touch panel thread created and lower level drivers opened successfully.
SSP_ERR_ASSERTION	A pointer parameter was NULL, or a lower level driver reported this error.
SSP_ERR_INTERNAL	The touch panel thread or event flags could not be created, or a lower level driver reported this error.
SSP_ERR_ALREADY_OPEN	Touch panel framework is already configured.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sf_touch_panel_chip_api_t::open
- sf_touch_panel_chip_api_t::reset

Note

This function is reentrant for any panel.

Store user configurations in control block.

Create a mutex to protect access to the control structure and the lower level hardware.

Open the lower level driver.

Delete RTOS services used and log the error

Reset the touch chip.

Delete RTOS services used and log the error

Create event flags for internal communication.

Create semaphore for use with touchDataGet function

Delete RTOS services used

Create main touch panel thread.

Delete RTOS services used

Mark control block open so other tasks know it is valid

◆ SF_TOUCH_PANEL_V2_Reset()

ssp_err_t SF_TOUCH_PANEL_V2_Reset (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*)

Implements sf_touch_panel_v2_api_t::reset.

Return values

SSP_SUCCESS	Touch chip reset successful.
SSP_ERR_ASSERTION	Parameter p_api_ctrl was NULL, or a lower level driver reported this error.
SSP_ERR_IN_USE	Mutex was not available, or a lower level driver reported this error.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Use Open API to configure.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- sf_touch_panel_chip_api_t::reset

Note

This function is reentrant for any panel.

Obtain mutex since this accesses shared resources.

Call hardware specific reset function.

Release mutex.

◆ SF_TOUCH_PANEL_V2_Start()

ssp_err_t SF_TOUCH_PANEL_V2_Start (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*)

Implements sf_touch_panel_v2_api_t::start.

Return values

SSP_SUCCESS	Enabled touch panel thread to scan for new touch events.
SSP_ERR_ASSERTION	A pointer parameter was NULL.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.

Note

This function is reentrant for any panel.

Set start flag.

◆ SF_TOUCH_PANEL_V2_Stop()

ssp_err_t SF_TOUCH_PANEL_V2_Stop (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*)

Implements sf_touch_panel_v2_api_t::stop.

Return values

SSP_SUCCESS	Disabled touch panel thread from scanning the touch events.
SSP_ERR_ASSERTION	A pointer parameter was NULL.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.

Note

This function is reentrant for any panel.

Set stop flag.

◆ SF_TOUCH_PANEL_V2_TouchDataGet()

ssp_err_t SF_TOUCH_PANEL_V2_TouchDataGet (sf_touch_panel_v2_ctrl_t *const *p_api_ctrl*, sf_touch_panel_v2_payload_t * *p_payload*, ULONG const *timeout*)

Implements sf_touch_panel_v2_api_t::touchDataGet.

Return values

SSP_SUCCESS	Touch panel data read successfully.
SSP_ERR_ASSERTION	A pointer parameter was NULL.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_V2_Open.
SSP_ERR_TIMEOUT	Time out occurs while waiting for semaphore.
SSP_ERR_WAIT_ABORTED	Suspension was aborted by another thread.

Note

This function is reentrant for any panel.
The function will pend until new touch event data is available or it will timeout.

◆ SF_TOUCH_PANEL_V2_VersionGet()

ssp_err_t SF_TOUCH_PANEL_V2_VersionGet (ssp_version_t *const *p_version*)
Implements sf_touch_panel_v2_api_t::versionGet. **Return values** SSP_SUCCESS — Version returned successfully. SSP_ERR_ASSERTION — Parameter p_version was null.

sf_touch_panel_v2_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Touch Panel V2 Framework

#include <sf_touch_panel_v2.h>

Public Member Functions

uint8_t stack [SF_TOUCH_PANEL_V2_STACK_SIZE]	BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)

Data Fields

uint32_t	open Used by driver to check if control block is valid.
uint16_t	hsize_pixels Horizontal size of screen in pixels.
uint16_t	vsize_pixels Vertical size of screen in pixels.
sf_touch_panel_v2_payload_t	payload Pointer to buffer used to store payload.
TX_MUTEX	mutex

	Mutex used to protect access to shared resources.
TX_EVENT_FLAGS_GROUP	flags Event flags for internal communication.
TX_THREAD	thread Main touch panel thread.
TX_SEMAPHORE	semaphore Semaphore used for SF_TOUCH_PANEL_V2_TouchDataGet.
sf_touch_panel_chip_instance_t const *	p_chip Pointer to touch chip.
uint16_t	update_hz
uint16_t	rotation_angle Touch coordinate rotation angle(0/90/180/270)
void const *	p_context Pointer to user callback context data.
bool	calibrate Used to check if calibration is required or not.

Detailed Description

Instance control block. DO NOT INITIALIZE. Initialization occurs when sf_touch_panel_v2_api_t::open is called

Member Function Documentation

◆ BSP_ALIGN_VARIABLE_V2()

uint8_t stack [SF_TOUCH_PANEL_V2_STACK_SIZE] sf_touch_panel_v2_instance_ctrl_t::BSP_ALIGN_VARIABLE_V2 (BSP_STACK_ALIGNMENT)
Stack for touch panel thread

Field Documentation

◆ update_hz

uint16_t sf_touch_panel_v2_instance_ctrl_t::update_hz
The frequency to report repeat (SF_TOUCH_PANEL_V2_EVENT_DOWN or SF_TOUCH_PANEL_V2_EVENT_HOLD) touch events in Hertz. ***Note*** *This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.*

The documentation for this struct was generated from the following file:

- sf_touch_panel_v2.h

5.1.3.35 UART Framework Instance

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Communications Framework UART implementation. More...

Data Structures

struct	sf_uart_comms_instance_ctrl_t
struct	sf_uart_comms_cfg_t

Macros

#define	SF_UART_COMMS_CODE_VERSION_MAJOR (2U)

Enumerations

enum	sf_uart_comms_state_t { SF_UART_COMMS_STATE_CLOSED = 0, SF_UART_COMMS_STATE_OPENED, SF_UART_COMMS_STATE_READING, SF_UART_COMMS_STATE_WRITING }

Functions

ssp_err_t	SF_UART_COMMS_Open (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_cfg_t const *const p_cfg) Open the UART for communication. More...
ssp_err_t	SF_UART_COMMS_Close (sf_comms_ctrl_t *const p_api_ctrl) Close the UART Channel and clean up the resources. More...
ssp_err_t	SF_UART_COMMS_Read (sf_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout) Read user specified number of bytes into destination buffer pointer. More...
ssp_err_t	SF_UART_COMMS_Write (sf_comms_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout) Write user specified number of bytes from the source buffer. More...
ssp_err_t	SF_UART_COMMS_Lock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type, UINT timeout) Lock the UART resource. More...
ssp_err_t	SF_UART_COMMS_Unlock (sf_comms_ctrl_t *const p_api_ctrl, sf_comms_lock_t lock_type) UnLock the UART resource. More...
ssp_err_t	SF_UART_COMMS_VersionGet (ssp_version_t *const p_version)

Detailed Description

RTOS-integrated Communications Framework UART implementation.

Macro Definition Documentation

◆ SF_UART_COMMS_CODE_VERSION_MAJOR

#define SF_UART_COMMS_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sf_uart_comms_state_t

enum sf_uart_comms_state_t	
Framework UART state	
Enumerator	
SF_UART_COMMS_STATE_CLOSED	UART port is closed.
SF_UART_COMMS_STATE_OPENED	UART port is opened.
SF_UART_COMMS_STATE_READING	UART port is on data reception.
SF_UART_COMMS_STATE_WRITING	UART port is on data transmission.

Function Documentation

◆ SF_UART_COMMS_Close()

ssp_err_t SF_UART_COMMS_Close (sf_comms_ctrl_t *const *p_api_ctrl*)

Close the UART Channel and clean up the resources.

Parameters

[in]	p_api_ctrl	Pointer to the UART control block

Return values

SSP_SUCCESS	UART channel is successfully closed.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_ctrl->p_lower_lvl_uart is NULL • Pointer p_ctrl->p_lower_lvl_uart->p_api is NULL • Pointer p_ctrl->p_lower_lvl_uart->p_api->close is NULL
SSP_ERR_NOT_OPEN	Channel is not opened.

Note

This function is reentrant for any channel.

Checks error. Further parameter checking can be done at the driver layer.

Calls close function of UART HAL driver

Release ThreadX resources

◆ SF_UART_COMMS_Lock()

ssp_err_t SF_UART_COMMS_Lock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*, UINT *timeout*)

Lock the UART resource.

Parameters

[in]	p_api_ctrl	Pointer to the UART control block
[in]	lock_type	Type of Lock.
[in]	timeout	timeout for lock.

Return values

SSP_SUCCESS	Locking UART resource successful.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_TIMEOUT	Timeout Error. 'Receive mutex get' timed out 'Transmit mutex get' timed out

Get both lock if requested, else get the lock type requested

◆ SF_UART_COMMS_Open()

ssp_err_t SF_UART_COMMS_Open (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_cfg_t const *const *p_cfg*)

Open the UART for communication.

Parameters

[in,out]	p_api_ctrl	Pointer to the UART control block
[in]	p_cfg	Pointer to the configuration structure

Return values

SSP_SUCCESS	Channel opened successfully.
SSP_ERR_IN_USE	Channel already in use.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_cfg is NULL • Pointer p_cfg->p_extend is NULL • Pointer p_cfg_extend->p_lower_lvl_uart is NULL • Pointer p_cfg_extend->p_lower_lvl_uart->p_api->open is NULL • Pointer p_cfg_extend->p_lower_lvl_uart->p_cfg is NULL
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

Returns

See Common Error Codes or functions called by this function for other possible codes. This function calls:

- uart_api_t::open

Note

This function is reentrant for any channel. Handle must be cleared by caller before calling this function.

Checks error. Further parameter checking can be done at the driver layer.

Initialize and start ThreadX resources

Calls open function of UART HAL driver

◆ SF_UART_COMMS_Read()

ssp_err_t SF_UART_COMMS_Read (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, UINT const *timeout*)

Read user specified number of bytes into destination buffer pointer.

Parameters

[in]	p_api_ctrl	Pointer to the UART control block
[in]	bytes	No.of bytes to be read
[in]	timeout	timeout for read
[out]	p_dest	Destination buffer

Return values

SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_ASSERTION	Pointer to UART control block/pointer to destination address is NULL.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode.
SSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
SSP_ERR_OVERFLOW	Hardware overflow.
SSP_ERR_FRAMING	Framing error.
SSP_ERR_PARITY	Parity error.
SSP_ERR_BREAK_DETECT	Break signal detected.
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Receive mutex get' timed out
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- uart_api_t::read
- uart_api_t::communicationAbort

Locks the UART reception hardware resource

Unlock the UART reception hardware resource

◆ SF_UART_COMMS_Unlock()

ssp_err_t SF_UART_COMMS_Unlock (sf_comms_ctrl_t *const *p_api_ctrl*, sf_comms_lock_t *lock_type*)

UnLock the UART resource.

Parameters

[in]	p_api_ctrl	Pointer to the UART control block
[in]	lock_type	Type of Lock.

Return values

SSP_SUCCESS	Unlocking UART resource successful.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_INTERNAL	Failed to release the mutex.

◆ SF_UART_COMMS_VersionGet()

ssp_err_t SF_UART_COMMS_VersionGet (ssp_version_t *const *p_version*)

Return values

SSP_SUCCESS	Version number obtained successfully.
SSP_ERR_ASSERTION	Pointer to version is NULL

◆ SF_UART_COMMS_Write()

ssp_err_t SF_UART_COMMS_Write (sf_comms_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, UINT const *timeout*)

Write user specified number of bytes from the source buffer.

Parameters

[in]	p_api_ctrl	Pointer to the UART control block
[in]	bytes	Number of bytes to be written.
[in]	timeout	Timeout for write. This timeout must be long enough for the write to complete. A timeout of 0 or

		TX_NO_WAIT results in a return value of SSP_ERR_TIMEOUT.
[out]	p_src	Source buffer

Return values

SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Pointer to source buffer is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set in handle.
SSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.
SSP_ERR_HW_LOCKED	Could not lock hardware.
SSP_ERR_TIMEOUT	'Transmit mutex get' timed out
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- uart_api_t::write
- uart_api_t::communicationAbort

Locks the UART transmission hardware resource

Clear the event flag.

Calls write function of UART HAL driver

Wait until write operation is completed. Event is signaled in event flag object

Calls communicationAbort function of UART HAL driver to abort write operation.

Unlock the UART transmission hardware resource

sf_uart_comms_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » UART Framework Instance

#include <sf_uart_comms.h>

Data Fields

uint32_t	state UART status.
uart_instance_t const *	p_lower_lvl_uart Pointer to UART interface (copied from cfg)
TX_MUTEX	mutex [2] Pointer to the mutex object for UART resource mutual exclusion.
TX_EVENT_FLAGS_GROUP	eventflag [2] Pointer to the event flag object for UART data transfer.
TX_QUEUE	queue Queue for reading.
uint32_t	queue_mem [SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS] Queue memory.

Detailed Description

UART communications instance control structure. DO NOT INITIALIZE. Initialization occurs when sf_comms_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_uart_comms.h

sf_uart_comms_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » UART Framework Instance

#include <sf_uart_comms.h>

Data Fields

uart_instance_t const *	p_lower_lvl_uart

Detailed Description

Configuration for RTOS integrated UART driver

Field Documentation

◆ p_lower_lvl_uart

uart_instance_t const* sf_uart_comms_cfg_t::p_lower_lvl_uart
Pointer to UART Driver instance

The documentation for this struct was generated from the following file:

- sf_uart_comms.h

5.1.3.36 NetX Synergy Port

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated NetX Ethernet driver for the Renesas Synergy software and Synergy Ethernet IP.
More...

Modules

	Interface file between SF_EL_NX and PHY driver

Data Structures

struct	EMAC_BD
struct	nx_mac_address_t
struct	NX_CALLBACK_REC
struct	sf_el_nx_cfg_t
struct	NX_REC

Macros

#define	SF_EL_NX_API_VERSION_MAJOR (2U)
#define	SF_EL_NX_CODE_VERSION_MAJOR (2U)
#define	MAX_ENET_INSTANCES 2

#define	TX_QUEUE_DEPTH 10
#define	NX_ENET_MAX_MTU 1518U
#define	NX_INITIALIZE_DONE 0x0002U
#define	NX_LINK_ENABLED 0x0004U
#define	DESCRIPTOR_FLAG_ACTIVE 0x80000000U /* TACT/RACT - Tx/Rx descriptor active */

Functions

void	edmac_eint_isr (void) edmac_eint_isr More...
UINT	nx_synergy_ethernet_init (NX_REC *nx_rec_ptr, sf_el_nx_cfg_t *sf_el_nx_cfg_ptr, bool hw_padding) nx_synergy_ethernet_init More...
void	nx_driver_event_handler (NX_REC *nx_rec_ptr) nx_driver_event_handler More...
void	enet_hw_enable_interrupt (NX_REC *nx_rec_ptr) enet_hw_enable_interrupt More...
UINT	nx_synergy_ethernet_deinit (NX_REC *nx_rec_ptr, sf_el_nx_cfg_t *sf_el_nx_cfg_ptr) nx_synergy_ethernet_deinit More...
ssp_err_t	nx_ether_custom_packet_send (NX_PACKET_POOL *pool_ptr, NX_REC *nx_record_ptr, UCHAR *data, UINT length, USHORT ether_type, nx_mac_address_t dest_mac_address) nx_ether_custom_packet_send More...
ssp_err_t	nx_ether_driver (NX_IP_DRIVER *driver_req_ptr, NX_REC *nx_rec_ptr, sf_el_nx_cfg_t *sf_el_nx_cfg_ptr) nx_ether_driver More...
void	nx_ether_interrupt (NX_REC *nx_rec_ptr)

	nx_ether_interrupt More...
ssp_err_t	nx_ethernet_version_get (ssp_version_t *const p_version) Retrieve the API version number. More...

Detailed Description

RTOS-integrated NetX Ethernet driver for the Renesas Synergy software and Synergy Ethernet IP.

Macro Definition Documentation

◆ DESCRIPTOR_FLAG_ACTIVE

```
#define DESCRIPTOR_FLAG_ACTIVE   0x80000000U /* TACT/RACT - Tx/Rx descriptor active */
```

Bits in the status word of the tx / rx descriptors

◆ MAX_ENET_INSTANCES

```
#define MAX_ENET_INSTANCES   2
```

Determine if the driver uses IP deferred processing or direct ISR processing.

◆ NX_ENET_MAX_MTU

```
#define NX_ENET_MAX_MTU   1518U
```

Max Ethernet packet size (14+ 1500 +2)

◆ NX_INITIALIZE_DONE

```
#define NX_INITIALIZE_DONE   0x0002U
```

Device is Initialized

◆ NX_LINK_ENABLED

```
#define NX_LINK_ENABLED   0x0004U
```

Device is enabled

◆ SF_EL_NX_API_VERSION_MAJOR

#define SF_EL_NX_API_VERSION_MAJOR (2U)
Version of the API defined in this file

◆ SF_EL_NX_CODE_VERSION_MAJOR

#define SF_EL_NX_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ TX_QUEUE_DEPTH

#define TX_QUEUE_DEPTH 10
Netx packets queued before dropping

Function Documentation

◆ edmac_eint_isr()

void edmac_eint_isr (void)
edmac_eint_isr This is the Ethernet interrupt service routine. It clears the interrupt flag and calls nx_ether_interrupt to process the interrupt. Save context if RTOS is used Process the interrupt. Clear pending interrupt flag to make sure it doesn't fire again after exiting. Restore context if RTOS is used

◆ enet_hw_enable_interrupt()

void enet_hw_enable_interrupt (NX_REC * *nx_rec_ptr*)

enet_hw_enable_interrupt

This function enables interrupts for the given Ethernet port.

Parameters

[in]	nx_rec_ptr	: Pointer to Ethernet record structure.

Note

The pointer parameter passed to this function is already validated at the higher level.

Enable interrupts at the NVIC.

Enable interrupts at the Ethernet controller

◆ nx_driver_event_handler()

void nx_driver_event_handler (NX_REC * *nx_rec_ptr*)

nx_driver_event_handler

This function is called from the IP thread deferred event. On every deferred event, this routine checks for Phy link status and handles link up/down and link change. During initialization this routine is responsible for checking for autonegotiation.

Parameters

[in]	nx_rec_ptr	: Pointer to Ethernet record structure.

Determine previous link status.

Save link status and changed polling interval.

Check PHY link status.

Save link status and changed polling interval.

◆ nx_ether_custom_packet_send()

ssp_err_t nx_ether_custom_packet_send (NX_PACKET_POOL * *pool_ptr*, NX_REC * *nx_record_ptr*, UCHAR * *data*, UINT *length*, USHORT *ether_type*, nx_mac_address_t *dest_mac_address*)

nx_ether_custom_packet_send

Ethernet driver routine to send a user data in raw Ethernet packet frame from a Source HW address to destination HW address with the requested EtherType through a Ethernet channel

Parameters

[in]	pool_ptr	: Pointer to packet pool to use
[in]	nx_record_ptr	: Pointer to Ethernet record structure
[in]	data	: Pointer to data to be send
[in]	length	: length of data to send
[in]	ether_type	: Type of Ethernet packet
[in]	dest_mac_address	: Destination hardware address to send the user data

Return values

SSP_SUCCESS	: Call successful
SSP_ERR_ASSERTION	: - A parameter pointers point to NULL • Invalid Ethernet record structure pointer • Invalid data length
SSP_ERR_OUT_OF_MEMORY	: No memory is free to allocate the packet in the pool buffer
SSP_ERR_NOT_OPEN	: Link not enabled
SSP_ERR_INTERNAL	: - Maximum transmit queue depth has been reached • No packet is available to append the data

Free the packet that we will not send.

Setup the prepend pointer in order to build the Ethernet frame. Backup another 2 bytes to get 32-bit word alignment.

Build the actual Ethernet frame.

Build the sender's MAC access.

Set the Ethernet frame type.

Endian swapping if necessary.

Put the frame on the wire.

◆ nx_ether_driver()

ssp_err_t nx_ether_driver (NX_IP_DRIVER * *driver_req_ptr*, NX_REC * *nx_rec_ptr*, sf_el_nx_cfg_t * *sf_el_nx_cfg_ptr*)

nx_ether_driver

Ethernet driver function for Renesas Synergy.

Parameters

[in]	driver_req_ptr	: Pointer to driver request structure
[in]	nx_rec_ptr	: Pointer to Ethernet record structure
[in]	sf_el_nx_cfg_ptr	: sf_el_nx configuration structure pointer. This is similar to SSP configuration structure

Return values

SSP_SUCCESS	: Call successful.
SSP_ERR_ASSERTION	: One or more pointers point to NULL.

Returns

See NX user manual for all possible return values.

Perform parameter checking

Setup the IP pointer from the driver request.

Default to successful return.

Extract driver command.

Process the driver request.

Save the callback record pointer in the record structure.

Record the interface structure in the driver record.

This command is used in multi homed devices - return NX_SUCCESS by default.

Process driver initialization.

Initialize BDs.

Record the interface structure in the driver record.

Update associated channel information in driver record

Configure mac address.

Get MAC address from user.

Detect S5D5 Mask Rev 02 to enable software padding for it

Initialize the ETHERC and E-DMAC hardware.

Setup the link maximum IP layer transfer unit.

See if we can honor the NX_LINK_ENABLE request. NX_ALREADY_ENABLED: Device has already been enabled

Enable the interrupts, at the interrupt controller and Ethernet controller.

Make sure we are in the right state to do the NX_LINK_DISABLE.

Disable receive and transmit.

Clear link enabled flag.

Clear the enabled flag since there is no-one else.

Free the packet that we will not send.

Process driver send packet. Place the Ethernet frame at the front of the packet.

Adjust the prepend pointer to accommodate Ethernet header.

Check if there is enough space in packet to append data.

Packet underflow.

Free the packet as there is not enough space to append data in it.

Adjust packet length.

Setup the prepend pointer in order to build the Ethernet frame. Backup another 2 bytes to get 32-bit word alignment.

Build the actual Ethernet frame.

Use MAC broadcast for this frame.

Build the sender's MAC access.

Set the Ethernet frame type.

Endian swapping if necessary.

Put the frame on the wire.

Enable multicast.

The device automatically receives multicast packets. Nothing needs to be done.

Process driver deferred requests. Process a device driver function on behalf of the IP thread.

Un-Initialize the ETHERC, E-DMAC and EPTPC hardware

Return the unhandled command status.

Return NULL in the supplied return pointer.

◆ nx_ether_interrupt()

void nx_ether_interrupt (NX_REC * *nx_rec_ptr*)

nx_ether_interrupt

This function is the main Renesas Ethernet interrupt handler.

Parameters

[in]	nx_rec_ptr	: Pointer to Ethernet record structure

Read EDMAC and EtherC status register.

Clear all interrupts.

Frame transmit completed interrupt The MAC sets Transmit Complete flag only if all frames are transmitted. This creates a delay in processing transmitted frames. The work around: this driver process transmitted packets when frame receive interrupt occurs. This way the transmitted packets can be released before processing received packets, reducing the delay.

Frame received.

Special case for ECI interrupt + link change.

To report link status changes to the application, add a semaphore put or event flag set to the statement below.

Link present.

◆ nx_ethernet_version_get()

ssp_err_t nx_ethernet_version_get (ssp_version_t *const *p_version*)

Retrieve the API version number.

Return values

SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

◆ nx_synergy_ethernet_deinit()

UINT nx_synergy_ethernet_deinit (NX_REC * *nx_rec_ptr*, sf_el_nx_cfg_t * *sf_el_nx_cfg_ptr*)

nx_synergy_ethernet_deinit

This function performs reset of the ethernet controller and PTP controller.

Parameters

[in]	nx_rec_ptr	: Pointer to Ethernet record structure.
[in]	sf_el_nx_cfg_ptr	: sf_el_nx configuration structure pointer

Return values

NX_SUCCESS	: Call successful
NX_NOT_SUCCESSFUL	: Call not successful

Disable transmission and reception at the Ethernet controller

Reset the EPTPC, Resetting after detection of an erroneous flag

Reset the ETHERC and EDMAC

Wait for 64 PCLKA cycles to reset ETHERC and EDMAC

Double check the Interrupts are disabled

Disable interrupts at the NVIC.

Check for the ether channel

Hold PHY in Reset for channel 1

Hold PHY in Reset for channel 0

Disable clock to the module

◆ nx_synergy_ethernet_init()

UINT nx_synergy_ethernet_init (NX_REC * *nx_rec_ptr*, sf_el_nx_cfg_t * *sf_el_nx_cfg_ptr*, bool *hw_padding*)

nx_synergy_ethernet_init

This function initializes the specified Ethernet port.

Parameters

[in]	nx_rec_ptr	Pointer to Ethernet record structure.
[in]	hw_padding	Flag to indicate hardware 2 byte padding is used or not.
[in]	sf_el_nx_cfg_ptr	Pointer to configuration structure.

Return values

SSP_SUCCESS	Call successful.
NX_NOT_SUCCESSFUL	Call not successful.

Configure the Ethernet interrupt.

Enable clock

Reset PHY

Initialize & reset EDMAC and ETHERC

Wait at least 64 cycles of PCLKA to reset the EDMAC and ETHERC. PCLKA must be at least 12.5 MHz to use Ethernet, so wait at least 5.12 us.

Set ETHERC default modes 100 Mbps, Full duplex

Set to little Endian.

Initialize controller

Set up descriptor addresses

Configure FIFO

Enable EDMAC receive

Enable receive, transmit, ECI interrupts.

Initialize PHY.

Start PHY auto negotiation

Create a timer to poll for completion of autonegotiation.

Interface file between SF_EL_NX and PHY driver

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

Data Structures

struct	phy_record_t

Macros

#define	R_PHY_OK (int16_t)(0) PHY device is initialized successfully.
#define	R_PHY_ERROR (int16_t)(-1) PHY device is not initialized successfully.
#define	R_PHY_NORMAL_PREAMBLE 0x20U Standard preamble length (32bit 1's)
#define	R_PHY_SUPPRESSED_PREAMBLE 0x01U Preamble suppression value.
#define	PHY_MII_READ (0x2U << 12) PHY read OP code.
#define	PHY_MII_WRITE (0x1U << 12) PHY write OP code.
#define	PHY_REG_CONTROL (0x0000U) Basic Control register.
#define	PHY_REG_STATUS (0x0001U) Basic Status register.
#define	PHY_REG_IDENTIFIER1 (0x0002U) PHY Identifier 1 register.
#define	PHY_REG_IDENTIFIER2 (0x0003U)

	PHY Identifier 2 register.
#define	PHY_REG_AN_ADVERTISEMENT (0x0004U) Auto-Negotiation Advertisement register.
#define	PHY_REG_AN_LINK_PARTNER (0x0005U) Auto-Negotiation Link Partner Ability register.
#define	PHY_REG_AN_EXPANSION (0x0006U) Auto-Negotiation Expansion register.

Enumerations

enum	linkstat_t {
	PHY_NO_LINK = 0, PHY_LINK_10H, PHY_LINK_10F, PHY_LINK_100H,
	PHY_LINK_100F
	}

Detailed Description

Enumeration Type Documentation

◆ linkstat_t

enum linkstat_t	
Standard PHY speed and duplex operation mode	
Enumerator	
PHY_NO_LINK	Link not established.
PHY_LINK_10H	10Mbps Half duplex
PHY_LINK_10F	10Mbps Full duplex
PHY_LINK_100H	100Mbps Half duplex
PHY_LINK_100F	100Mbps Full duplex

phy_record_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port » Interface file between SF_EL_NX and PHY driver

#include <sf_el_nx.h>

Data Fields

uint16_t	preamble_length Preamble length.
uint16_t	local_advertise The capabilities of the local link as PHY data.

Detailed Description

Standard PHY data structure

The documentation for this struct was generated from the following file:

- sf_el_nx.h

EMAC_BD Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

#include <nx_renesas_synergy.h>

Data Fields

uint32_t	bd_status Status - 32 bits.
uint16_t	bd_rxdatalength data length
uint16_t	bd_bufsize Buffer size.
uint8_t *	bd_buffer_ptr

	Buffer pointer.
NX_PACKET *	bd_nx_packet Padding used to associate descriptor with nx_packet.

Detailed Description

Descriptor structure

The documentation for this struct was generated from the following file:

- nx_renesas_synergy.h

nx_mac_address_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

#include <nx_renesas_synergy.h>

Detailed Description

MAC Address structure

The documentation for this struct was generated from the following file:

- nx_renesas_synergy.h

NX_CALLBACK_REC Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

#include <nx_renesas_synergy.h>

Detailed Description

SF_EL_NX callback record structure

The documentation for this struct was generated from the following file:

- nx_renesas_synergy.h

sf_el_nx_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

#include <nx_renesas_synergy.h>

Detailed Description

sf_el_nx configuration structure is similar to SSP configuration structure. This is collection of parameters for this module that is required for an instance initialization.

The documentation for this struct was generated from the following file:

- nx_renesas_synergy.h

NX_REC Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » NetX Synergy Port

#include <nx_renesas_synergy.h>

Data Fields

UINT	nx_state state of this driver
NX_IP *	ip_ptr NetX IP structure handling this controller.
uint16_t	channel Channel associated with this record.
UCHAR	rx_bd_space [sizeof(EMAC_BD) *(NUM_RX_DESC+1)]
EMAC_BD *	driver_rx_bd Pointer to Receive BD.

EMAC_BD *	driver_tx_bd Pointer to Transmit BD.
ULONG	driver_tx_bd_index Transmit BD buffer index.
ULONG	driver_tx_bd_in_use Number of transmit BD in use.
ULONG	driver_tx_release_index Index of Transmit buffer index to release.
ULONG	driver_rx_bd_index Receive BD buffer index.
UINT	driver_packets_queued [not used]
NX_PACKET *	driver_tx_packet_queue [not used]
NX_PACKET *	driver_tx_packet_queue_end [not used]
R_ETHERC0_Type *	etherc_ptr Pointer to ETHERC register base address.
R_EDMAC0_Type *	edmac_ptr Pointer to EDMAC register base address.
UINT	link_established link status flag

UINT	nx_driver_phy_polling_requested Polling flag.
NX_CALLBACK_REC *	p_callback_rec pointer to the user callback function
IRQn_Type	irq Interrupt priority number.

Detailed Description

Driver record structure is similar to SSP control structure that describes an open instance of sf_el_nx module.

Field Documentation

◆ rx_bd_space

UCHAR NX_REC::rx_bd_space[sizeof(EMAC_BD) *(NUM_RX_DESC+1)]
Allocate space for the BDs. Build one extra BD there because the BDs must be 16 byte aligned.

The documentation for this struct was generated from the following file:

- nx_renesas_synergy.h

5.1.3.37 NetX Synergy Port PHY Driver

Renesas Synergy Software Package Reference » Framework Layer

Interface between SF_EL_NX Ethernet framework and PHY driver. More...

Functions

int16_t	bsp_ethernet_phy_init (uint32_t channel) bsp_ethernet_phy_init - Initialize Ethernet PHY device. More...
void	bsp_ethernet_phy_start_autonegotiate (uint32_t channel, uint8_t pause)

	Sets Auto-Negotiation advertisement and starts auto-negotiation. More...
int16_t	bsp_ethernet_phy_get_autonegotiate (uint32_t channel, uint16_t *p_line_speed_duplex, uint16_t *p_local_pause, uint16_t *p_partner_pause) Gets capabilities of an Ethernet PHY device. More...
int16_t	bsp_ethernet_phy_get_link_status (uint32_t channel) bsp_ethernet_phy_get_link_status - Returns the status of the physical link. More...

Detailed Description

Interface between SF_EL_NX Ethernet framework and PHY driver.

Function Documentation

◆ bsp_ethernet_phy_get_autonegotiate()

int16_t bsp_ethernet_phy_get_autonegotiate (uint32_t *channel*, uint16_t * *p_line_speed_duplex*, uint16_t * *p_local_pause*, uint16_t * *p_partner_pause*)

Gets capabilities of an Ethernet PHY device.

Parameters

[in]	channel	Ethernet channel number
[in]	p_line_speed_duplex	A pointer to the location of both the line speed and the duplex
[in]	p_local_pause	A pointer to the location to store the local pause bits
[in]	p_partner_pause	A pointer to the location to store the partner pause bits

Return values

R_PHY_OK	Got information successfully
R_PHY_ERROR	PHY device is yet to be initialized

Note

Validating parameters is not required by this function as it by design is only called by sf_el_nx framework with valid parameters.

Reads the status register. Because reading the first time shows the previous state, the Link status bit should be read twice.

Checks the link status

Check the auto-negotiation status

Gets local pause capability

◆ bsp_ethernet_phy_get_link_status()

int16_t bsp_ethernet_phy_get_link_status (uint32_t *channel*)

bsp_ethernet_phy_get_link_status - Returns the status of the physical link.

Parameters

[in]	channel	Ethernet channel number

Return values

R_PHY_OK	PHY device is initialized successfully
R_PHY_ERROR	PHY device is not initialized successfully

◆ bsp_ethernet_phy_init()

int16_t bsp_ethernet_phy_init (uint32_t *channel*)

bsp_ethernet_phy_init - Initialize Ethernet PHY device.

Parameters

[in]	channel	Ethernet channel number

Return values

R_PHY_OK	PHY device is initialized successfully
R_PHY_ERROR	PHY device is not initialized successfully or Validation of PHY chip fails.

Note

Refer HW manual for valid channels for a target MCU

Read the PHY Identifier register and compare read value with the PHY chip OUI identifier number

Resets PHY device

Waits the reset completion, PHY_CONTROL_RESET bit is self-cleared after 1 is written to it.

When MICREL_KSZ8091RNB of the Micrel, Inc. is used, the pin that outputs the state of LINK is used combinedly with ACTIVITY in default. The setting of the pin is changed so that only the state of LINK is output. Set Clock Mode to 50MHz

Sets Duplex Mode as Full-duplex

◆ bsp_ethernet_phy_start_autonegotiate()

void bsp_ethernet_phy_start_autonegotiate (uint32_t *channel*, uint8_t *pause*)

Sets Auto-Negotiation advertisement and starts auto-negotiation.

Parameters

[in]	channel	Ethernet channel number
[in]	pause	Using state of pause frames

5.1.3.38 BLE Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated BLE Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces: More...

Data Structures

struct	sf_ble_on_rl78g1d_cfg_t

Macros

#define	SF_BLE_RL78G1D_CODE_VERSION_MAJOR (2U)
#define	SF_BLE_RL78G1D_CODE_VERSION_MINOR (0U)

Functions

ssp_err_t	SF_BLE_RL78G1D_Open (sf_ble_ctrl_t *const p_ctrl, sf_ble_cfg_t const *const p_cfg) Initialize BLE RL78G1D driver Implements sf_ble_api_t::open Following tasks are performed Initialize module Store user specified configuration in internal driver data for further use. More...
ssp_err_t	SF_BLE_RL78G1D_Close (sf_ble_ctrl_t *const p_ctrl) De-Initialize BLE driver Implements sf_ble_api_t::close Following tasks are performed Terminate working threads and timers Reset internal driver data. More...
ssp_err_t	SF_BLE_RL78G1D_InfoGet (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_info_t *const p_ble_info) Get BLE module information Implements sf_ble_api_t::infoGet Gets module information like chip-set information and RSSI value. More...
ssp_err_t	SF_BLE_RL78G1D_ProvisioningGet (sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t *const p_ble_provisioning) Reads the current BLE Provisioning information. More...
ssp_err_t	SF_BLE_RL78G1D_ProvisioningSet (sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t const *const p_ble_provisioning) Provisions the RL78G1D BLE Driver. More...
ssp_err_t	SF_BLE_RL78G1D_Scan (sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t *const p_scan, uint8_t *const p_cnt, sf_ble_scan_info_t *p_scan_info) Scans for available BLE devices. More...
ssp_err_t	SF_BLE_RL78G1D_AdvertisementStart (sf_ble_ctrl_t *const p_ctrl, sf_ble_adv_info_t *const p_adv_info) Make the device discoverable Implements

	sf_ble_api_t::advertisementStart Broadcasts device information to make it discoverable for other BLE modules. More...
ssp_err_t	SF_BLE_RL78G1D_AdvertisementStop (sf_ble_ctrl_t *const p_ctrl) Stop the device from being discoverable Implements sf_ble_api_t::advertisementStop Stop broadcasting device information to other BLE modules. More...
ssp_err_t	SF_BLE_RL78G1D_WhitelistAdd (sf_ble_ctrl_t *const p_ctrl, uint8_t const *const p_bd_addr) Add specified devices to whitelist Implements sf_ble_api_t::whitelistAdd Adds the devices to whitelist. More...
ssp_err_t	SF_BLE_RL78G1D_WhitelistDel (sf_ble_ctrl_t *const p_ctrl, uint8_t const *const p_bd_addr) Remove specified devices from whitelist Implements sf_ble_api_t::whitelistDel Removes devices from whiteList. More...
ssp_err_t	SF_BLE_RL78G1D_BondingStart (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint8_t const *const p_bd_addr, sf_ble_bonding_start_t *p_bonding_start) Initiate bonding process with remote BLE device Implements sf_ble_api_t::bondingStart Initiates bonding process and exchange security keys if enabled. More...
ssp_err_t	SF_BLE_RL78G1D_BondingResponse (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint8_t const *const p_bd_addr, sf_ble_bonding_response_t *p_bonding_resp) Respond to the bonding request from the remote BLE device Implements sf_ble_api_t::bondingResponse Send bonding response on reception of SF_BLE_EVENT_BONDING_INDICATION. More...
ssp_err_t	SF_BLE_RL78G1D_StartEncryption (sf_ble_ctrl_t *const p_ctrl, sf_ble_sm_enc_info_t const *p_enc_info) Start encryption with remote BLE device Implements sf_ble_api_t::startEncryption Encrypts with remote BLE device. More...
ssp_err_t	SF_BLE_RL78G1D_Connect (sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle) Connect to a remote BLE device Implements sf_ble_api_t::connect

	Initiate a connection with remote BLE device. More...
ssp_err_t	SF_BLE_RL78G1D_Listen (sf_ble_ctrl_t *const p_ctrl) Listen for connection request from remote device Implements sf_ble_api_t::listen. More...
ssp_err_t	SF_BLE_RL78G1D_Authorization (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle) Specifies remote device has been authorized by user Implements sf_ble_api_t::authorization Indicates that the specified remote device has been authorized by user. More...
ssp_err_t	SF_BLE_RL78G1D_Disconnect (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle) Terminate connection with remote BLE device Implements sf_ble_api_t::disconnect Disconnects with remote BLE device. More...
ssp_err_t	SF_BLE_RL78G1D_SetTxPower (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info) Sets the transmit power for the procedure specified by the connection handle Implements sf_ble_api_t::setTxPower Sets the transmit power for the procedure. Valid values of TX power level (p_tx_power_info->power_lvl) for RL78G1D in dBm are -15, -10, -7, -2, -1 and 0. More...
ssp_err_t	SF_BLE_RL78G1D_AddCustomProfiles (sf_ble_ctrl_t *const p_ctrl, sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len, sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len) Add GATT Attributes to create Custom GATT Profile Implements sf_ble_api_t::gattAddCustomProfiles This function is called with list of service and characteristics which is to be added to GATT database. Services and characteristics which were previously added will be removed and newly passed services and characteristics will be added. More...
ssp_err_t	SF_BLE_RL78G1D_GATTServiceDiscovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt) Perform service discovery used by GATT client Implements sf_ble_api_t::gattServiceDiscovery GATT client performs service discovery of GATT server. More...

ssp_err_t	SF_BLE_RL78G1D_GATTCharDiscovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt) Perform characteristics discovery used by GATT client Implements sf_ble_api_t::gattCharDiscovery GATT client performs characteristics discovery of GATT server. More...
ssp_err_t	SF_BLE_RL78G1D_GATTCharDescDiscovery (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle, sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t *const p_rsp_cnt) Perform characteristics descriptor discovery used by GATT client Implements sf_ble_api_t::gattCharDescDiscovery GATT client performs characteristics descriptor discovery of GATT server. More...
ssp_err_t	SF_BLE_RL78G1D_GATTCharRead (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp) Perform read characteristic used by GATT client Implements sf_ble_api_t::gattCharRead GATT client reads from GATT server. More...
ssp_err_t	SF_BLE_RL78G1D_GATTCharWrite (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const p_char_write_req) Perform write characteristic used by GATT client Implements sf_ble_api_t::gattCharWrite GATT client performs write operation on GATT server. More...
ssp_err_t	SF_BLE_RL78G1D_GATTCharExecuteWrite (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag) Perform execute write on all pending write operations, used by GATT client Implements sf_ble_api_t::gattCharExecuteWrite GATT client performs execute write on all pending write operations on GATT server. More...
ssp_err_t	SF_BLE_RL78G1D_GATTCharWriteLocal (sf_ble_ctrl_t *const p_ctrl, uint16_t char_handle, uint16_t data_length, uint8_t *const p_data) Perform local characteristic write used by GATT server Implements sf_ble_api_t::gattCharWriteLocal GATT server performs local characteristic write. More...

ssp_err_t	SF_BLE_RL78G1D_GATTSendNotify (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send notification to GATT client, used by GATT server Implements sf_ble_api_t::gattSendNotify GATT server sends notification to GATT client. More...
ssp_err_t	SF_BLE_RL78G1D_GATTSendIndicate (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle) Send indication to GATT client, used by GATT server Implements sf_ble_api_t::gattSendIndicate GATT server sends indication to GATT client. More...
ssp_err_t	SF_BLE_RL78G1D_GATTWriteResponse (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t error_code) Send response to write operation received by GATT client, used by GATT server Implements sf_ble_api_t::gattWriteResponse GATT server sends response of write operation to GATT client. More...
ssp_err_t	SF_BLE_RL78G1D_VersionGet (ssp_version_t *const p_version) Get driver version based on compile time macros. Implements sf_ble_api_t::versionGet. More...

Detailed Description

RTOS-integrated BLE Framework example. Implementation of RL78G1D BLE Driver. It implements the following interfaces:

- SF BLE Framework Interface

Macro Definition Documentation

◆ SF_BLE_RL78G1D_CODE_VERSION_MAJOR

```
#define SF_BLE_RL78G1D_CODE_VERSION_MAJOR   (2U)
```

BLE Interface. Major Version of code that implements the API defined in this file

◆ SF_BLE_RL78G1D_CODE_VERSION_MINOR

```
#define SF_BLE_RL78G1D_CODE_VERSION_MINOR   (0U)
```

Minor Version of code that implements the API defined in this file

Function Documentation

◆ SF_BLE_RL78G1D_AddCustomProfiles()

ssp_err_t SF_BLE_RL78G1D_AddCustomProfiles (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_svc_attribute_t * *p_svc_attr*, uint32_t *svc_attr_len*, sf_ble_char_attribute_t * *p_char_attr*, uint32_t *char_attr_len*)

Add GATT Attributes to create Custom GATT Profile Implements sf_ble_api_t::gattAddCustomProfiles This function is called with list of service and characteristics which is to be added to GATT database. Services and characteristics which were previously added will be removed and newly passed services and characteristics will be added.

If services and characteristics were previously added and now those services and characteristics are to be removed and no new services and characteristics are to be added , then call this API with service and characteristics length as zero

Return values

SSP_ERR_UNSUPPORTED	Unsupported Feature

◆ SF_BLE_RL78G1D_AdvertisementStart()

ssp_err_t SF_BLE_RL78G1D_AdvertisementStart (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_adv_info_t *const *p_adv_info*)

Make the device discoverable Implements sf_ble_api_t::advertisementStart Broadcasts device information to make it discoverable for other BLE modules.

Return values

SSP_SUCCESS	Start the advertisement of device successfully
SSP_ERR_BLE_FAILED	Advertisement Start failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_INVALID_MODE	Invalid GAP Role

◆ SF_BLE_RL78G1D_AdvertisementStop()

ssp_err_t SF_BLE_RL78G1D_AdvertisementStop (sf_ble_ctrl_t *const *p_ctrl*)

Stop the device from being discoverable Implements sf_ble_api_t::advertisementStop Stop broadcasting device information to other BLE modules.

Return values

SSP_SUCCESS	Stop the advertisement of device successfully
SSP_ERR_BLE_FAILED	Advertisement Stop failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_Authorization()

ssp_err_t SF_BLE_RL78G1D_Authorization (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*)

Specifies remote device has been authorized by user Implements sf_ble_api_t::authorization Indicates that the specified remote device has been authorized by user.

Return values

SSP_SUCCESS	Authorization success
SSP_ERR_BLE_FAILED	Authorization failed
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_BondingResponse()

ssp_err_t SF_BLE_RL78G1D_BondingResponse (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint8_t const *const *p_bd_addr*, sf_ble_bonding_response_t * *p_bonding_resp*)

Respond to the bonding request from the remote BLE device Implements sf_ble_api_t::bondingResponse Send bonding response on reception of SF_BLE_EVENT_BONDING_INDICATION.

Return values

SSP_SUCCESS	Bonding response success
SSP_ERR_BLE_FAILED	Failed getting bonding response
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_BondingStart()

ssp_err_t SF_BLE_RL78G1D_BondingStart (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint8_t const *const *p_bd_addr*, sf_ble_bonding_start_t * *p_bonding_start*)

Initiate bonding process with remote BLE device Implements sf_ble_api_t::bondingStart Initiates bonding process and exchange security keys if enabled.

Return values

SSP_SUCCESS	Bonding start success
SSP_ERR_BLE_FAILED	Bonding start failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_Close()

ssp_err_t SF_BLE_RL78G1D_Close (sf_ble_ctrl_t *const *p_ctrl*)

De-Initialize BLE driver Implements sf_ble_api_t::close Following tasks are performed Terminate working threads and timers Reset internal driver data.

Return values

SSP_SUCCESS	Suspend the driver functionality
SSP_ERR_BLE_FAILED	Close failure
SSP_ERR_NOT_OPEN	Module is not opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_Connect()

ssp_err_t SF_BLE_RL78G1D_Connect (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_connection_t const *const *p_conn*, sf_ble_conn_handle_t * *p_handle*)

Connect to a remote BLE device Implements sf_ble_api_t::connect Initiate a connection with remote BLE device.

Return values

SSP_SUCCESS	Connect success
SSP_ERR_BLE_FAILED	Failed to connect
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_INVALID_MODE	Invalid GAP Role

◆ SF_BLE_RL78G1D_Disconnect()

ssp_err_t SF_BLE_RL78G1D_Disconnect (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*)

Terminate connection with remote BLE device Implements sf_ble_api_t::disconnect Disconnects with remote BLE device.

Return values

SSP_SUCCESS	Disconnect success
SSP_ERR_BLE_FAILED	Disconnect failed
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharDescDiscovery()

ssp_err_t SF_BLE_RL78G1D_GATTCharDescDiscovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *start_handle*, uint16_t *end_handle*, sf_ble_char_desc_discovery_rsp_t *const *p_sf_ble_chardesc_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

Perform characteristics descriptor discovery used by GATT client Implements sf_ble_api_t::gattCharDescDiscovery GATT client performs characteristics descriptor discovery of GATT server.

Return values

SSP_ERR_BLE_FAILED	Characteristics descriptor discovery failed
SSP_SUCCESS	Characteristics descriptor discovery successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharDiscovery()

ssp_err_t SF_BLE_RL78G1D_GATTCharDiscovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_discovery_req_t const *const *p_sf_ble_char_dscv_req*, sf_ble_char_discovery_rsp_t *const *p_sf_ble_char_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

Perform characteristics discovery used by GATT client Implements sf_ble_api_t::gattCharDiscovery GATT client performs characteristics discovery of GATT server.

Return values

SSP_ERR_BLE_FAILED	Characteristics discovery failed
SSP_SUCCESS	Characteristics discovery successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharExecuteWrite()

ssp_err_t SF_BLE_RL78G1D_GATTCharExecuteWrite (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_execute_write_t *execute_flag*)

Perform execute write on all pending write operations, used by GATT client Implements sf_ble_api_t::gattCharExecuteWrite GATT client performs execute write on all pending write operations on GATT server.

Return values

SSP_ERR_BLE_FAILED	Execute write failed
SSP_SUCCESS	Execute write successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharRead()

ssp_err_t SF_BLE_RL78G1D_GATTCharRead (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_read_req_t const *const *p_char_read_req*, sf_ble_char_read_rsp_t *const *p_char_read_rsp*)

Perform read characteristic used by GATT client Implements sf_ble_api_t::gattCharRead GATT client reads from GATT server.

Return values

SSP_ERR_BLE_FAILED	Characteristic read failed
SSP_SUCCESS	Characteristic read successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharWrite()

ssp_err_t SF_BLE_RL78G1D_GATTCharWrite (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_char_write_req_t const *const *p_char_write_req*)

Perform write characteristic used by GATT client Implements sf_ble_api_t::gattCharWrite GATT client performs write operation on GATT server.

Return values

SSP_ERR_BLE_FAILED	Characteristic write failed
SSP_SUCCESS	Characteristic write successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTCharWriteLocal()

ssp_err_t SF_BLE_RL78G1D_GATTCharWriteLocal (sf_ble_ctrl_t *const *p_ctrl*, uint16_t *char_handle*, uint16_t *data_length*, uint8_t *const *p_data*)

Perform local characteristic write used by GATT server Implements sf_ble_api_t::gattCharWriteLocal GATT server performs local characteristic write.

Return values

SSP_ERR_BLE_FAILED	Local characteristic write failed
SSP_SUCCESS	Local characteristic write successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTSendIndicate()

ssp_err_t SF_BLE_RL78G1D_GATTSendIndicate (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t *p_handle*, uint16_t *char_handle*)

Send indication to GATT client, used by GATT server Implements sf_ble_api_t::gattSendIndicate GATT server sends indication to GATT client.

Return values

SSP_ERR_BLE_FAILED	Send Indication failed
SSP_SUCCESS	Send Indication successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTSendNotify()

ssp_err_t SF_BLE_RL78G1D_GATTSendNotify (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *char_handle*)

Send notification to GATT client, used by GATT server Implements sf_ble_api_t::gattSendNotify GATT server sends notification to GATT client.

Return values

SSP_ERR_BLE_FAILED	Send notification failed
SSP_SUCCESS	Send notification successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTServiceDiscovery()

ssp_err_t SF_BLE_RL78G1D_GATTServiceDiscovery (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_service_discovery_req_t const *const *p_sf_ble_svc_dscv_req*, sf_ble_service_discovery_rsp_t *const *p_sf_ble_svc_dscv_rsp*, uint32_t *const *p_rsp_cnt*)

Perform service discovery used by GATT client Implements sf_ble_api_t::gattServiceDiscovery GATT client performs service discovery of GATT server.

Return values

SSP_ERR_BLE_FAILED	Service discovery failed
SSP_SUCCESS	Service discovery successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_GATTWriteResponse()

ssp_err_t SF_BLE_RL78G1D_GATTWriteResponse (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, uint16_t *handle*, sf_ble_attribute_error_code_t *error_code*)

Send response to write operation received by GATT client, used by GATT server Implements sf_ble_api_t::gattWriteResponse GATT server sends response of write operation to GATT client.

Return values

SSP_ERR_BLE_FAILED	Send write response failed
SSP_SUCCESS	Send write response successful
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_InfoGet()

ssp_err_t SF_BLE_RL78G1D_InfoGet (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t * *p_handle*, sf_ble_info_t *const *p_ble_info*)

Get BLE module information Implements sf_ble_api_t::infoGet Gets module information like chip-set information and RSSI value.

Return values

SSP_SUCCESS	Successfully get the BLE information
SSP_ERR_BLE_FAILED	Info get failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_TIMEOUT	BLE event timeout

Initialize RSSI to zero and do not check return value of Read RSSI API since InfoGet API can be called before connection is established and in that case Read RSSI API will return error but InfoGet API should return Success with 0 RSSI

◆ SF_BLE_RL78G1D_Listen()

ssp_err_t SF_BLE_RL78G1D_Listen (sf_ble_ctrl_t *const *p_ctrl*)

Listen for connection request from remote device Implements sf_ble_api_t::listen.

Return values

SSP_ERR_UNSUPPORTED	Functionality is not supported.

◆ SF_BLE_RL78G1D_Open()

ssp_err_t SF_BLE_RL78G1D_Open (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_cfg_t const *const *p_cfg*)

Initialize BLE RL78G1D driver Implements sf_ble_api_t::open Following tasks are performed Initialize module Store user specified configuration in internal driver data for further use.

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_BLE_FAILED	Failed to initialize module
SSP_ERR_ALREADY_OPEN	Device is already open
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_INTERNAL	GATT Initialization failure

If mutex is already created then mutex create returns TX_MUTEX_ERROR which is a valid condition

◆ SF_BLE_RL78G1D_ProvisioningGet()

ssp_err_t SF_BLE_RL78G1D_ProvisioningGet (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_provisioning_t *const *p_ble_provisioning*)

Reads the current BLE Provisioning information.

Implements sf_ble_api_t::provisionGet Reads the provisioning information

Return values

SSP_SUCCESS	Successfully reads provisioning information
SSP_ERR_BLE_FAILED	Provisioning get failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_ProvisioningSet()

ssp_err_t SF_BLE_RL78G1D_ProvisioningSet (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_provisioning_t const *const *p_ble_provisioning*)

Provisions the RL78G1D BLE Driver.

Implements sf_ble_api_t::provisionSet This function performs the following tasks: Provisions the BLE driver. Set bonding and security modes as provisioned.

Return values

SSP_SUCCESS	Successfully provisioned the device.
SSP_ERR_BLE_FAILED	Provisioning set failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_UNSUPPORTED	Unsupported feature

◆ SF_BLE_RL78G1D_Scan()

ssp_err_t SF_BLE_RL78G1D_Scan (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_scan_t *const *p_scan*, uint8_t *const *p_cnt*, sf_ble_scan_info_t * *p_scan_info*)

Scans for available BLE devices.

Implements sf_ble_api_t::scan Scans for available BLE devices and return the list to caller.

Return values

SSP_SUCCESS	Successful scan of available BLE devices
SSP_ERR_BLE_FAILED	Scan failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_INVALID_MODE	Invalid GAP Role

◆ SF_BLE_RL78G1D_SetTxPower()

ssp_err_t SF_BLE_RL78G1D_SetTxPower (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_conn_handle_t *const *p_handle*, sf_ble_set_tx_pwr_info_t * *p_tx_power_info*)

Sets the transmit power for the procedure specified by the connection handle Implements sf_ble_api_t::setTxPower Sets the transmit power for the procedure. Valid values of TX power level (p_tx_power_info->power_lvl) for RL78G1D in dBm are -15, -10, -7, -2, -1 and 0.

Return values

SSP_SUCCESS	TX power set successfully
SSP_ERR_BLE_FAILED	TX power set failed
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments
SSP_ERR_INVALID_ARGUMENT	Invalid transmit power level parameter not supported by device

◆ SF_BLE_RL78G1D_StartEncryption()

ssp_err_t SF_BLE_RL78G1D_StartEncryption (sf_ble_ctrl_t *const *p_ctrl*, sf_ble_sm_enc_info_t const * *p_enc_info*)

Start encryption with remote BLE device Implements sf_ble_api_t::startEncryption Encrypts with remote BLE device.

Return values

SSP_SUCCESS	in case of success
SSP_ERR_BLE_FAILED	Failed to start encryption
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_VersionGet()

ssp_err_t SF_BLE_RL78G1D_VersionGet (ssp_version_t *const *p_version*)

Get driver version based on compile time macros. Implements sf_ble_api_t::versionGet.

Return values

SSP_SUCCESS	Success
SSP_ERR_ASSERTION	Argument NULL is passed

◆ SF_BLE_RL78G1D_WhitelistAdd()

ssp_err_t SF_BLE_RL78G1D_WhitelistAdd (sf_ble_ctrl_t *const *p_ctrl*, uint8_t const *const *p_bd_addr*)

Add specified devices to whitelist Implements sf_ble_api_t::whitelistAdd Adds the devices to whitelist.

Return values

SSP_SUCCESS	Whitelist add success
SSP_ERR_BLE_FAILED	Whitelist add failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

◆ SF_BLE_RL78G1D_WhitelistDel()

ssp_err_t SF_BLE_RL78G1D_WhitelistDel (sf_ble_ctrl_t *const *p_ctrl*, uint8_t const *const *p_bd_addr*)

Remove specified devices from whitelist Implements sf_ble_api_t::whitelistDel Removes devices from whiteList.

Return values

SSP_SUCCESS	Whitelist delete success
SSP_ERR_BLE_FAILED	Whitelist delete failure
SSP_ERR_NOT_OPEN	Device Not Opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_TIMEOUT	BLE event timeout
SSP_ERR_ASSERTION	Invalid Arguments

sf_ble_on_rl78g1d_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » BLE Framework on RL78G1D

#include <sf_ble_rl78g1d.h>

Data Fields

sf_comms_instance_t const *	p_low_lvl_sf_comms Pointer to Communication Framework instance.
timer_instance_t const *	p_low_lvl_timer Pointer to timer instance.
UINT	ble_driver_thread_priority BLE Driver Thread Priority.
UINT	ble_serial_thread_priority BLE Serial Thread Priority.

Detailed Description

Extension structure for this Implementation. Each implementation can have its own extension structure. For example one implementation may use RSPI for communicating with the BLE chip while another may use SDIO.

The documentation for this struct was generated from the following file:

- sf_ble_rl78g1d.h

5.1.3.39 BLE On-Board Profile Framework on RL78G1D

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Driver. It implements the following interfaces: More...

Macros

#define	SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MAJOR	(2U)
#define	SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MINOR	(0U)

Detailed Description

RTOS-integrated BLE On-Board Profile Framework example. Implementation of RL78G1D BLE On-Board Profile Driver. It implements the following interfaces:

- SF BLE On-Board Profile Framework Interface

Macro Definition Documentation

◆ **SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MAJOR**

#define SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MAJOR (2U)
BLE On-Board Profile Interface. Major Version of code that implements the API defined in this file

◆ **SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MINOR**

#define SF_BLE_ONBOARD_PROFILE_RL78G1D_CODE_VERSION_MINOR (0U)
Minor Version of code that implements the API defined in this file

5.1.3.40 Cellular Framework Example using Quectel CATM1

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Cellular Framework example. Implementation of Cellular Quectel CATM1 Driver. It implements the following interfaces: More...

Data Structures

struct	sf_cellular_qctlcatm1_extended_cfg_t

Macros

#define	SF_CELLULAR_QCTLCATM1_CODE_VER_MAJOR (2U)
#define	SF_CELLULAR_QCTLCATM1_CODE_VER_MINOR (0U)

Enumerations

enum	sf_cellular_qctlcatm1_network_scan_seq_t { SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATM1_LTECATNB1_GSM = 0, SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATM1_GSM_LTECATNB1 = 1, SF_CELLULAR_QCTLCATM1_NWSCANSEQ_GSM_LTECATNB1_LTECATM1 = 3, SF_CELLULAR_QCTLCATM1_NWSCANSEQ_GSM_LTECATM1_LTECATNB1 = 2,

```
                                SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATNB1_LTECATM1_GSM = 4,
                                SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATNB1_GSM_LTECATM1 = 5
                                }
```

Detailed Description

RTOS-integrated Cellular Framework example. Implementation of Cellular Quectel CATM1 Driver. It implements the following interfaces:

- SF CELLULAR Framework Interface

Macro Definition Documentation

◆ SF_CELLULAR_QCTLCATM1_CODE_VER_MAJOR

```
#define SF_CELLULAR_QCTLCATM1_CODE_VER_MAJOR   (2U)
```

Cellular Interface. Major Version of code that implements the API defined in this file

◆ SF_CELLULAR_QCTLCATM1_CODE_VER_MINOR

```
#define SF_CELLULAR_QCTLCATM1_CODE_VER_MINOR   (0U)
```

Minor Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sf_cellular_qctlcatm1_network_scan_seq_t

enum sf_cellular_qctlcatm1_network_scan_seq_t	
Cellular Fallback sequence type	
Enumerator	
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATM1_LTECATNB1_GSM	Network scan sequence Default (LTE Cat.M1->LTE Cat.NB1->GSM)
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATM1_GSM_LTECATNB1	Network scan sequence LTE Cat.M1->GSM->LTE Cat.NB1.
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_GSM_LTECATNB1_LTECATM1	Network scan sequence GSM->LTE Cat.NB1->LTE Cat.M1.
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_GSM_LTECATM1_LTECATNB1	Network scan sequence GSM->LTE Cat.M1->LTE Cat.NB1.
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATNB1_LTECATM1_GSM	Network scan sequence LTE Cat.NB1->LTE Cat.M1->GSM.
SF_CELLULAR_QCTLCATM1_NWSCANSEQ_LTECATNB1_GSM_LTECATM1	Network scan sequence LTE Cat.NB1->GSM->LTE Cat.M1.

sf_cellular_qctlcatm1_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Cellular Framework Example using Quectel CATM1

#include <sf_cellular_qctlcatm1.h>

Data Fields

sf_cellular_qctlcatm1_network_scan_seq_t	nwscanseq Network fall back sequence selection.
uint8_t *	nbiot_band_selection NBIOT band selections.

Detailed Description

Extended configuration for Quectel CATM1 BG96

The documentation for this struct was generated from the following file:

- sf_cellular_qctlcatm1.h

5.1.3.41 BSD Socket over Quectel CATM1 on-chip stack

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Cellular Socket Framework example. Implementation of Quectel CATM1 Socket layer over Quectel CATM1 On-Chip stack It implements the following interfaces: More...

RTOS-integrated Cellular Socket Framework example. Implementation of Quectel CATM1 Socket layer over Quectel CATM1 On-Chip stack It implements the following interfaces:

- SF Socket CELLULAR Framework Interface

5.1.3.42 Cellular Framework Example using RYZ014 CATM1

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Cellular Framework example. Implementation of Cellular RYZ014 CATM1 Driver. It implements the following interfaces: More...

Macros

#define	SF_CELLULAR_RYZ014CATM1_CODE_VER_MAJOR	(2U)
#define	SF_CELLULAR_RYZ014CATM1_CODE_VER_MINOR	(0U)

Detailed Description

RTOS-integrated Cellular Framework example. Implementation of Cellular RYZ014 CATM1 Driver. It implements the following interfaces:

- SF CELLULAR Framework Interface

Macro Definition Documentation

◆ SF_CELLULAR_RYZ014CATM1_CODE_VER_MAJOR

#define SF_CELLULAR_RYZ014CATM1_CODE_VER_MAJOR (2U)
Cellular Interface. Major Version of code that implements the API defined in this file

◆ SF_CELLULAR_RYZ014CATM1_CODE_VER_MINOR

#define SF_CELLULAR_RYZ014CATM1_CODE_VER_MINOR (0U)
Minor Version of code that implements the API defined in this file

5.1.3.43 BSD Socket over RYZ014CATM1 on-chip stack

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated Cellular Socket Framework example. Implementation of RYZ014CATM1 Socket layer over RYZ014CATM1 On-Chip stack It implements the following interfaces: More...

RTOS-integrated Cellular Socket Framework example. Implementation of RYZ014CATM1 Socket layer over RYZ014CATM1 On-Chip stack It implements the following interfaces:

- SF Socket CELLULAR Framework Interface

5.1.3.44 Touch Panel Framework Example for FT5X06

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated touch panel chip ft5x06 example. Implementation of ft5x06 touch chip Driver. It implements the following interfaces: More...

Data Structures

struct	sf_touch_panel_chip_ft5x06_instance_ctrl_t
struct	sf_touch_panel_chip_on_ft5x06_cfg_t

Detailed Description

RTOS-integrated touch panel chip ft5x06 example. Implementation of ft5x06 touch chip Driver. It implements the following interfaces:

- Touch chip Interface

sf_touch_panel_chip_ft5x06_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Touch Panel Framework Example for FT5X06

#include <sf_touch_panel_chip_ft5x06.h>

Data Fields

ioport_port_pin_t	pin Reset pin.
sf_i2c_instance_t const *	p_lower_lvl_framewrk Pointer to lower level I2C framework.
sf_external_irq_instance_t const *	p_lower_lvl_irq Pointer to lower level external IRQ.
sf_touch_panel_v2_payload_t	last_payload Stores most recent payload for comparison.

Detailed Description

Instance control block. DO NOT INITIALIZE. Initialization occurs when sf_touch_panel_chip_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_ft5x06.h

sf_touch_panel_chip_on_ft5x06_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Touch Panel Framework Example for FT5X06

#include <sf_touch_panel_chip_ft5x06.h>

Data Fields

ioport_port_pin_t	pin Port pin connected to reset line on touch controller chip.
sf_i2c_instance_t const *	p_lower_lvl_framewrk Pointer to lower level I2C framework.
sf_external_irq_instance_t const *	p_lower_lvl_irq Pointer to lower level external IRQ.

Detailed Description

Configuration for RTOS touch panel driver.

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_ft5x06.h

5.1.3.45 Touch Panel Framework Example for SX8654

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated touch panel chip sx8654 example. Implementation of sx8654 touch chip Driver. It implements the following interfaces: More...

Data Structures

struct	sf_touch_panel_chip_sx8654_instance_ctrl_t
struct	sf_touch_panel_chip_on_sx8654_cfg_t

Detailed Description

RTOS-integrated touch panel chip sx8654 example. Implementation of sx8654 touch chip Driver. It implements the following interfaces:

- Touch chip Interface

sf_touch_panel_chip_sx8654_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Touch Panel Framework Example for SX8654

#include <sf_touch_panel_chip_sx8654.h>

Data Fields

ioport_port_pin_t	pin Reset pin.
sf_i2c_instance_t const *	p_lower_lvl_framewrk Pointer to lower level I2C framework.
sf_external_irq_instance_t const *	p_lower_lvl_irq Pointer to lower level external IRQ.
uint16_t	hsize_pixels Horizontal size of screen in pixels.
uint16_t	vsize_pixels Vertical size of screen in pixels.
sf_touch_panel_v2_payload_t	last_payload Stores most recent payload for comparison.

Detailed Description

Instance control block. DO NOT INITIALIZE. Initialization occurs when sf_touch_panel_chip_api_t::open is called

---

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_sx8654.h

sf_touch_panel_chip_on_sx8654_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » Touch Panel Framework Example for SX8654

#include <sf_touch_panel_chip_sx8654.h>

Data Fields

ioport_port_pin_t	pin Port pin connected to reset line on touch controller chip.
sf_i2c_instance_t const *	p_lower_lvl_framewrk Pointer to lower level I2C framework.
sf_external_irq_instance_t const *	p_lower_lvl_irq Pointer to lower level external IRQ.
uint16_t	hsize_pixels Horizontal size of screen in pixels.
uint16_t	vsize_pixels Vertical size of screen in pixels.

Detailed Description

Configuration for RTOS touch panel driver.

The documentation for this struct was generated from the following file:

- sf_touch_panel_chip_sx8654.h

5.1.3.46 WiFi Framework on GT202

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated WiFi Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces: More...

Data Structures

struct	sf_wifi_on_gt202_cfg_t

Macros

#define	SF_WIFI_GT202_CODE_VERSION_MAJOR (2U)
#define	SF_WIFI_GT202_CODE_VERSION_MINOR (0U)

Functions

ssp_err_t	SF_WIFI_GT202_Open (sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const *const p_cfg) Initialize WiFi module. More...
ssp_err_t	SF_WIFI_GT202_Close (sf_wifi_ctrl_t *const p_ctrl) Stop WiFi module functionality. More...
ssp_err_t	SF_WIFI_GT202_ProvisioningSet (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t const *const p_wifi_provisioning) Provisions the WiFi module. More...
ssp_err_t	SF_WIFI_GT202_ProvisioningGet (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t *const p_wifi_provisioning) Reads the current WiFi Provisioning information of the WiFi module. More...
ssp_err_t	SF_WIFI_GT202_MulticastListAdd (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr) Add MAC address in multicast list. More...
ssp_err_t	SF_WIFI_GT202_MulticastListDelete (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr) Delete MAC address from multicast list. More...
ssp_err_t	SF_WIFI_GT202_StatisticsGet (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_stats_t *const p_wifi_device_stats) Get the interface statistics. More...

ssp_err_t	SF_WIFI_GT202_Transmit (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length) Transmit data packets. More...
ssp_err_t	SF_WIFI_GT202_InfoGet (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t *const p_wifi_info) Get WiFi module information. More...
ssp_err_t	SF_WIFI_GT202_Scan (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t *const p_scan, uint8_t *const p_cnt) Scans for available APs. More...
ssp_err_t	SF_WIFI_GT202_ACLAdd (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Add MAC address from Access control list. More...
ssp_err_t	SF_WIFI_GT202_ACLDelete (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Delete MAC address from Access control list. More...
ssp_err_t	SF_WIFI_GT202_MACAddressSet (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac) Set MAC address of WiFi module. More...
ssp_err_t	SF_WIFI_GT202_MACAddressGet (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_mac) Get MAC address of WiFi module. More...
ssp_err_t	SF_WIFI_GT202_WpsStart (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_wps_t const *const p_wps) Start WiFi WPS. More...
ssp_err_t	SF_WIFI_GT202_VersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. Implements sf_wifi_api_t::versionGet. More...

Detailed Description

RTOS-integrated WiFi Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces:

- SF WIFI Framework Interface

Macro Definition Documentation

◆ SF_WIFI_GT202_CODE_VERSION_MAJOR

#define SF_WIFI_GT202_CODE_VERSION_MAJOR (2U)
WiFi Interface. Major Version of code that implements the API defined in this file

◆ SF_WIFI_GT202_CODE_VERSION_MINOR

#define SF_WIFI_GT202_CODE_VERSION_MINOR (0U)
Minor Version of code that implements the API defined in this file

Function Documentation

◆ SF_WIFI_GT202_ACLAdd()

ssp_err_t SF_WIFI_GT202_ACLAdd (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*)
Add MAC address from Access control list. Implements sf_wifi_api_t::ACLAdd Add specified mac address in access control list **Return values** SSP_ERR_UNSUPPORTED — Functionality is not supported.

◆ SF_WIFI_GT202_ACLDelete()

ssp_err_t SF_WIFI_GT202_ACLDelete (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*)
Delete MAC address from Access control list. Implements sf_wifi_api_t::ACLDelete Delete specified mac address from access control list **Return values** SSP_ERR_UNSUPPORTED — Functionality is not supported.

◆ SF_WIFI_GT202_Close()

ssp_err_t SF_WIFI_GT202_Close (sf_wifi_ctrl_t *const *p_ctrl*)

Stop WiFi module functionality.

Implements sf_wifi_api_t::close This function performs the following tasks: Update global variables for future use. Disable the Interrupt and suspend the driver task thread.

Return values

SSP_SUCCESS	Suspend the driver functionality.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to close

Dis-associate or Stop Access Point

Stop WiFi Driver

Delete byte pool used by WiFi driver

Terminate and Delete WiFi Driver task thread

Set init done flag to false and driver handle to NULL

◆ SF_WIFI_GT202_InfoGet()

ssp_err_t SF_WIFI_GT202_InfoGet (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_info_t *const *p_wifi_info*)

Get WiFi module information.

Implements sf_wifi_api_t::infoGet Get WiFi module information like chipset/driver information, RSSI, noise level, link quality

Return values

SSP_SUCCESS	Successfully get the WiFi information
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_WIFI_FAILED	Failed reading WiFi information
SSP_ERR_IN_USE	Module in use

◆ SF_WIFI_GT202_MACAddressGet()

ssp_err_t SF_WIFI_GT202_MACAddressGet (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t *const *p_mac*)

Get MAC address of WiFi module.

Implements sf_wifi_api_t::getMACAddress Read configured MAC address of the WiFi module.

Return values

SSP_SUCCESS	Successfully reads the mac address.
SSP_ERR_WIFI_FAILED	Failed to read mac address
SSP_ERR_NOT_OPEN	Driver not opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Argument NULL is passed

Driver param structure for ioctl

◆ SF_WIFI_GT202_MACAddressSet()

ssp_err_t SF_WIFI_GT202_MACAddressSet (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*)

Set MAC address of WiFi module.

Implements sf_wifi_api_t::setMACAddress Configure MAC address of the WiFi module.

Return values

SSP_ERR_UNSUPPORTED	Functionality is not supported.

◆ SF_WIFI_GT202_MulticastListAdd()

ssp_err_t SF_WIFI_GT202_MulticastListAdd (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac_addr*)

Add MAC address in multicast list.

Implements sf_wifi_api_t::multicastListAdd Adds specified MAC address in Multicast list

Return values

SSP_ERR_UNSUPPORTED	Functionality is not supported by WiFi module

◆ SF_WIFI_GT202_MulticastListDelete()

ssp_err_t SF_WIFI_GT202_MulticastListDelete (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac_addr*)

Delete MAC address from multicast list.

Implements sf_wifi_api_t::multicastListDelete Deletes specified MAC Address from Multicast list

Return values

SSP_ERR_UNSUPPORTED	Functionality is not supported.

◆ SF_WIFI_GT202_Open()

ssp_err_t SF_WIFI_GT202_Open (sf_wifi_ctrl_t * *p_ctrl*, sf_wifi_cfg_t const *const *p_cfg*)

Initialize WiFi module.

Implements sf_wifi_api_t::open This function performs the following tasks: Initializes WiFi module and Configure the parameters as per the p_cfg Update global variables for future use.

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_ALREADY_OPEN	WiFi module is already opened
SSP_ERR_WIFI_CONFIG_FAILED	WiFi module Configuration failed
SSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to initialize
SSP_ERR_UNSUPPORTED	Unsupported Parameter configuration
SSP_ERR_INTERNAL	Extended Configuration is NULL

◆ SF_WIFI_GT202_ProvisioningGet()

ssp_err_t SF_WIFI_GT202_ProvisioningGet (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_provisioning_t *const *p_wifi_provisioning*)

Reads the current WiFi Provisioning information of the WiFi module.

Implements sf_wifi_api_t::provisioningGet Reads the provisioning information

Return values

SSP_SUCCESS	Successfully reads provisioning information
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_WIFI_FAILED	Failed to get provisioning information
SSP_ERR_ASSERTION	Argument NULL is passed

◆ SF_WIFI_GT202_ProvisioningSet()

ssp_err_t SF_WIFI_GT202_ProvisioningSet (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_provisioning_t const *const *p_wifi_provisioning*)

Provisions the WiFi module.

Implements sf_wifi_api_t::provisioningSet This function performs the following tasks: Provisions the WiFi driver. Start WiFi interface in AP or Client mode as provisioned.

Return values

SSP_SUCCESS	Successfully provisioned the driver.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_WIFI_FAILED	Failed to set provisioning configuration.
SSP_ERR_INVALID_SIZE	Invalid length of security key
SSP_ERR_INVALID_ARGUMENT	Invalid encryption type for given security
SSP_ERR_IN_USE	Module in use

if is_opened

◆ SF_WIFI_GT202_Scan()

ssp_err_t SF_WIFI_GT202_Scan (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_scan_t *const *p_scan*, uint8_t *const *p_cnt*)

Scans for available APs.

Implements sf_wifi_api_t::scan Scan for available AP's SSID and return the list to caller.

Return values

SSP_SUCCESS	Successfully scan the network for available APs
SSP_ERR_NOT_OPEN	Driver not opened
SSP_ERR_WIFI_FAILED	Failed to scan
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use

◆ SF_WIFI_GT202_StatisticsGet()

ssp_err_t SF_WIFI_GT202_StatisticsGet (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_stats_t *const *p_wifi_device_stats*)

Get the interface statistics.

Implements sf_wifi_api_t::statisticsGet Collect the statistics information of WiFi interface

Return values

SSP_SUCCESS	Successfully get the Statistics information.
SSP_ERR_UNSUPPORTED	Functionality is not supported.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed reading WiFi statistics

Statistics are not available in case of On chip Stack

◆ SF_WIFI_GT202_Transmit()

ssp_err_t SF_WIFI_GT202_Transmit (sf_wifi_ctrl_t *const *p_ctrl*, uint8_t *const *p_buf*, uint32_t *length*)

Transmit data packets.

Implements sf_wifi_api_t::transmit Adds packets in transmit Queue.

Return values

SSP_SUCCESS	Successfully added the packet in transmit queue
SSP_ERR_NOT_OPEN	WiFi driver is not opened
SSP_ERR_OUT_OF_MEMORY	Memory allocation failed
SSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to transmit
SSP_ERR_UNSUPPORTED	When using On-Chip stack

Transmit function will be used only when using NSAL

◆ SF_WIFI_GT202_VersionGet()

ssp_err_t SF_WIFI_GT202_VersionGet (ssp_version_t *const *p_version*)

Set driver version based on compile time macros. Implements sf_wifi_api_t::versionGet.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ SF_WIFI_GT202_WpsStart()

ssp_err_t SF_WIFI_GT202_WpsStart (sf_wifi_ctrl_t *const *p_ctrl*, sf_wifi_wps_t const *const *p_wps*)

Start WiFi WPS.

Implements sf_wifi_api_t::wpsStart Start WPS to connect device.

Return values

SSP_SUCCESS	WPS started Successfully and device is able to connect
SSP_ERR_INTERNAL	Internal error
SSP_ERR_WIFI_FAILED	Failed to connect WiFi module using WPS method
SSP_ERR_NOT_OPEN	Driver not opened
SSP_ERR_IN_USE	Module in use
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_INVALID_ARGUMENT	Invalid input parameters
SSP_ERR_WIFI_WPS_MULTIPLE_PB_SESSIONS	Another Push button session is already in progress
SSP_ERR_TIMEOUT	WPS Timer expired
SSP_ERR_WIFI_WPS_M2D_RECEIVED	M2D Error code received which means Registrar is unable to authenticate with the Enrollee
SSP_ERR_WIFI_WPS_AUTHENTICATION_FAILED	WPS authentication failed
SSP_ERR_WIFI_WPS_CANCELLED	WPS Request was not accepted by underlying driver
SSP_ERR_WIFI_WPS_INVALID_PIN	Invalid WPS Pin

sf_wifi_on_gt202_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » WiFi Framework on GT202

#include <sf_wifi_gt202.h>

Data Fields

spi_instance_t const *	p_lower_lvl_spi SPI Interface used for WiFi communications.

external_irq_instance_t const *	p_lower_lvl_icu ICU Interface used for WiFi communications.
ioport_port_pin_t	pin_reset Port pin used for resetting chipset.
ioport_port_pin_t	pin_slave_select Port pin used for SPI slave select.
uint32_t	driver_task_priority Driver task thread priority, should be high priority.

Detailed Description

Extension structure for this Implementation. Each implementation can have its own extension structure. For example one implementation may use RSPI for communicating with the WiFi chip while another may use SDIO.

The documentation for this struct was generated from the following file:

- sf_wifi_gt202.h

5.1.3.47 WiFi On Chip Stack on GT202

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated WiFi On Chip Stack Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces: More...

Macros

#define	SF_WIFI_GT202_ONCHIP_STACK_CODE_MAJOR (2U)
#define	SF_WIFI_GT202_ONCHIP_STACK_CODE_MINOR (0U)

Detailed Description

RTOS-integrated WiFi On Chip Stack Framework example. Implementation of Atheros WiFi Driver. It implements the following interfaces:

- SF WIFI On-Chip Stack Interface

Macro Definition Documentation

◆ **SF_WIFI_GT202_ONCHIP_STACK_CODE_MAJOR**

#define SF_WIFI_GT202_ONCHIP_STACK_CODE_MAJOR (2U)
WiFi Interface. Major Version of code that implements the API defined in this file

◆ **SF_WIFI_GT202_ONCHIP_STACK_CODE_MINOR**

#define SF_WIFI_GT202_ONCHIP_STACK_CODE_MINOR (0U)
Minor Version of code that implements the API defined in this file

5.1.3.48 BSD Socket on GT202

Renesas Synergy Software Package Reference » Framework Layer

Implementation of GT202 Socket layer over GT202 On-Chip stack. More...

Macros

#define	SF_WIFI_GT202_SOCKET_CODE_MAJOR (2U)
#define	SF_WIFI_GT202_SOCKET_CODE_MINOR (0U)
#define	AF_LOCAL (1) Local Socket Family.
#define	AF_INET (2) IPV4 Socket Family.
#define	AF_INET6 (10) IPV6 Socket Family.

Enumerations

enum	sf_wifi_socket_type_t { SOCK_STREAM = 0, SOCK_DGRAM, SOCK_STREAM = 1,

	SOCK_DGRAM, SOCK_RAW }

Variables

const sf_socket_api_t	g_sf_socket_wifi_on_sf_socket_wifi_gt202 Socket API interface.

Detailed Description

Implementation of GT202 Socket layer over GT202 On-Chip stack.

Macro Definition Documentation

◆ SF_WIFI_GT202_SOCKET_CODE_MAJOR

#define SF_WIFI_GT202_SOCKET_CODE_MAJOR (2U)
WiFi Interface. Major Version of code that implements the API defined in this file

◆ SF_WIFI_GT202_SOCKET_CODE_MINOR

#define SF_WIFI_GT202_SOCKET_CODE_MINOR (0U)
Minor Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sf_wifi_socket_type_t

enum sf_wifi_socket_type_t	
Type of Socket	
Enumerator	
SOCK_STREAM	TCP Socket.
SOCK_DGRAM	UDP Socket.
SOCK_STREAM	TCP Socket.
SOCK_DGRAM	UDP Socket.
SOCK_RAW	RAW Socket.

5.1.3.49 WiFi Framework on QCA4010

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated WiFi Framework example. Implementation of Silex ULPGN WiFi Driver. It implements the following interfaces: More...

Data Structures

struct	sf_wifi_qca4010_instance_cfg_t

Macros

#define	SF_WIFI_QCA4010_CODE_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_CODE_VERSION_MINOR (0U)

Functions

ssp_err_t	SF_WiFi_QCA4010_Open (sf_wifi_qca4010_ctrl_t *p_ctrl, sf_wifi_qca4010_cfg_t const *const p_cfg) Initialize WiFi module. More...
ssp_err_t	SF_WiFi_QCA4010_Close (sf_wifi_qca4010_ctrl_t *const p_ctrl) Stop Wifi QCA4010 driver functionality. More...
ssp_err_t	SF_WiFi_QCA4010_ProvisioningSet (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_provisioning_t const *const p_wifi_provisioning) Sets the provisioning information. More...
ssp_err_t	SF_WiFi_QCA4010_WifiStatusGet (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_status_t *const p_wifi_status) Get the network information. More...
ssp_err_t	SF_WiFi_QCA4010_Scan (sf_wifi_qca4010_ctrl_t *p_ctrl, sf_wifi_qca4010_scan_t *const p_scan, uint8_t count) Scans for available APs. More...
ssp_err_t	SF_Wifi_QCA4010_CommandSend (sf_wifi_qca4010_ctrl_t *const p_ctrl, sf_wifi_qca4010_cmd_resp_t *const p_input_at_command, sf_wifi_qca4010_cmd_resp_t *const p_output, uint32_t const timeout)

	Send custom AT command to SX-ULPGN module. More...
ssp_err_t	SF_WiFi_QCA4010_VersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. Implements sf_wifi_qca4010_api_t::versionGet. More...

Detailed Description

RTOS-integrated WiFi Framework example. Implementation of Silex ULPGN WiFi Driver. It implements the following interfaces:

- SF WIFI QCA4010 Framework Interface

Macro Definition Documentation

◆ SF_WIFI_QCA4010_CODE_VERSION_MAJOR

#define SF_WIFI_QCA4010_CODE_VERSION_MAJOR (2U)
WiFi Interface. Major Version of code that implements the API defined in this file

◆ SF_WIFI_QCA4010_CODE_VERSION_MINOR

#define SF_WIFI_QCA4010_CODE_VERSION_MINOR (0U)
Minor Version of code that implements the API defined in this file

Function Documentation

◆ SF_WiFi_QCA4010_Close()

ssp_err_t SF_WiFi_QCA4010_Close (sf_wifi_qca4010_ctrl_t *const *p_ctrl*)

Stop Wifi QCA4010 driver functionality.

Implements sf_wifi_qca4010_api_t::close De-initializes the lower level interface

Parameters

[in]	p_ctrl	Wifi control block

Return values

SSP_SUCCESS	Wifi Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_WIFI_FAILED	Driver De-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex

Disconnect from the access point

close module

Set module open flag to false and delete mutex

Release the mutex

Delete the mutex

◆ SF_Wifi_QCA4010_CommandSend()

ssp_err_t SF_Wifi_QCA4010_CommandSend (sf_wifi_qca4010_ctrl_t *const *p_ctrl*, sf_wifi_qca4010_cmd_resp_t *const *p_input_at_command*, sf_wifi_qca4010_cmd_resp_t *const *p_output*, uint32_t const *timeout*)

Send custom AT command to SX-ULPGN module.

This API will send AT command provided by user to the SX-ULPGN and will collect the response from the modem and will send it back to the user. If silex is in Data Mode when this API is called then Framework will first switch Modem to Command Mode, then send the AT command and collect the response and then switches the Modem back to Data Mode.

Parameters

[in]	p_ctrl	Pointer to the wifi control block
[in]	p_input_at_command	Pointer to structure which contains command to send
[in,out]	p_output	Pointer to buffer in which response will be sent to user, Also user will pass the size of the buffer which is pointed by p_output
[in]	timeout	Timeout for which framework will wait for response in milliseconds

Return values

SSP_SUCCESS	Successfully sent AT command and collected response
SSP_ERR_WIFI_FAILED	Failed to either send AT command or collect response
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use

Get Mutex

Send AT command

Release the mutex

◆ SF_WiFi_QCA4010_Open()

ssp_err_t SF_WiFi_QCA4010_Open (sf_wifi_qca4010_ctrl_t * *p_ctrl*, sf_wifi_qca4010_cfg_t const *const *p_cfg*)

Initialize WiFi module.

Implements sf_wifi_qca4010_api_t::open This function performs the following tasks: Initializes WiFi module and Configure the parameters as per the p_cfg Update global variables for future use.

Parameters

[out]	p_ctrl	Wifi control block
[in]	p_cfg	Wifi configuration structure

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_ALREADY_OPEN	WiFi module is already opened
SSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Module initialization failed
SSP_ERR_INTERNAL	Internal Error occurred

Create Mutex for Synchronization

Get Mutex Lock

Open and configure the wifi module

Release Mutex

◆ SF_WiFi_QCA4010_ProvisioningSet()

ssp_err_t SF_WiFi_QCA4010_ProvisioningSet (sf_wifi_qca4010_ctrl_t *const *p_ctrl*, sf_wifi_qca4010_provisioning_t const *const *p_wifi_provisioning*)

Sets the provisioning information.

Implements sf_wifi_qca4010_api_t::provisioningSet Sets Wifi's provisioning information

Parameters

[in]	p_ctrl	Wifi control block
[in]	p_wifi_provisioning	Wifi provisioning structure

Return values

SSP_SUCCESS	Successfully set the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_WIFI_FAILED	Provisioning failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_INTERNAL	Internal Error occurred
SSP_ERR_INVALID_ARGUMENT	Invalid input value or No commas are accepted in the SSID or key.

Get Mutex

Provision in AP or client mode

Release the mutex

◆ SF_WiFi_QCA4010_Scan()

ssp_err_t SF_WiFi_QCA4010_Scan (sf_wifi_qca4010_ctrl_t * *p_ctrl*, sf_wifi_qca4010_scan_t *const *p_scan*, uint8_t *count*)

Scans for available APs.

Implements sf_wifi_qca4010_api_t::scan Scan for available AP's SSID and return the list to caller.

Parameters

[in]	p_ctrl	Wifi control block
[out]	p_scan	Wifi scan structure
[in]	count	Number of access points to be scanned

Return values

SSP_SUCCESS	Successfully scan the network for available APs
SSP_ERR_NOT_OPEN	Driver not opened
SSP_ERR_WIFI_FAILED	Failed to scan
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex

Scan for available access points

Release the mutex

◆ SF_WiFi_QCA4010_VersionGet()

ssp_err_t SF_WiFi_QCA4010_VersionGet (ssp_version_t *const *p_version*)

Set driver version based on compile time macros. Implements sf_wifi_qca4010_api_t::versionGet.

Parameters

[in]	p_version	Wifi version

Return values

SSP_SUCCESS	Version information retrieved successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ SF_WiFi_QCA4010_WifiStatusGet()

ssp_err_t SF_WiFi_QCA4010_WifiStatusGet (sf_wifi_qca4010_ctrl_t *const *p_ctrl*, sf_wifi_qca4010_status_t *const *p_wifi_status*)

Get the network information.

Implements sf_wifi_qca4010_api_t::statisticsGet Collect the statistics information of WiFi interface

Parameters

[in]	p_ctrl	Wifi control block
[out]	p_wifi_status	Wifi status structure

Return values

SSP_SUCCESS	Successfully get the Statistics information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed reading WiFi statistics information
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex

Get Wifi Network information

Release the mutex

sf_wifi_qca4010_instance_cfg_t Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » WiFi Framework on QCA4010

#include <sf_wifi_qca4010.h>

Data Fields

uint32_t	num_uarts number of UARTS currently used for communication with module
uint8_t	is_opened Status flag storing framework open status.

uint8_t	init_done Status flag storing driver initialization status.
uint32_t	curr_cmd_port Current UART instance index for AT commands.
uint32_t	curr_data_port Current UART instance index for data.
volatile uint8_t	curr_socket_index Currently active socket instance.
sf_wifi_qca4010_cfg_t const *	p_cfg Instance configuration.
sf_wifi_qca4010_provisioning_t	prov_info Provisioning information.
TX_MUTEX *	p_wifi_mutex Mutex for Framework API synchronization.
uart_instance_t *	p_uart_instance_objects [SF_WIFI_QCA4010_CFG_MAX_NUMBER_UART_PORTS] UART instance objects.
TX_QUEUE	socket_queue [12] Queue to read data from socket.
uint8_t	command_flag Variable to indicate data or command mode.
uint8_t *	p_resp_buff Array to store AT command response.

uint32_t	resp_buffer_length Response buffer length.
uint8_t	is_data_mode_on Status flag storing data mode status.

Detailed Description

SF Wifi framework instance configuration

The documentation for this struct was generated from the following file:

- sf_wifi_qca4010.h

5.1.3.50 WiFi On Chip Stack on QCA4010

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated WiFi On Chip Stack Framework example. Implementation of SILEX WiFi Driver. It implements the following interfaces: More...

Macros

#define	SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MINOR (0U)

Functions

ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_Open (sf_wifi_qca4010_onchip_stack_ctrl_t *p_ctrl, sf_wifi_qca4010_onchip_stack_cfg_t const *const p_cfg) Open the WiFi Device driver to use the Onchip stack support. More...
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_Close (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl) Close the Wifi Device driver. More...
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_Ping (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG

	*p_ip_addr, uint32_t count, uint32_t interval_ms)
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_IPAddressConfigure (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_qca4010_onchip_stack_ip_cfg_t *const p_ip_cfg) Configure static IP or enable DHCP. More...
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_VersionGet (ssp_version_t *const p_version) This function gets the version information of the underlying driver. More...
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStart (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip) Starts DHCP server. More...
ssp_err_t	SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStop (sf_wifi_qca4010_onchip_stack_ctrl_t *const p_ctrl, ULONG *p_start_ip, ULONG *p_end_ip) Stops DHCP server. More...

Detailed Description

RTOS-integrated WiFi On Chip Stack Framework example. Implementation of SILEX WiFi Driver. It implements the following interfaces:

- SF WIFI QCA4010 On-Chip Interface

Macro Definition Documentation

◆ SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MAJOR

```
#define SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MAJOR  (2U)
```

WiFi Interface. Major Version of code that implements the API defined in this file

◆ SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MINOR

```
#define SF_WIFI_QCA4010_ONCHIP_CODE_VERSION_MINOR  (0U)
```

Minor Version of code that implements the API defined in this file

Function Documentation

◆ SF_WIFI_QCA4010_ONCHIP_STACK_Close()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_Close (sf_wifi_qca4010_onchip_stack_ctrl_t *const *p_ctrl*)

Close the Wifi Device driver.

Parameters

[in]	p_ctrl	Wifi control block Implements sf_wifi_qca4010_onchip_stack_api_t::close Calls the low level wifi device driver's Close API to close the wifi Driver.

Return values

SSP_SUCCESS	Wifi Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_WIFI_FAILED	Driver De-initialization failed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex

Close lower level wifi driver

Release the mutex

Delete Mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStart()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStart (sf_wifi_qca4010_onchip_stack_ctrl_t *const *p_ctrl*, ULONG * *p_start_ip*, ULONG * *p_end_ip*)

Starts DHCP server.

Implements sf_wifi_qca4010_onchip_stack_api_t::dhcpServerStart Starts DHCP server using on CHIP networking stack support.

Parameters

[in]	p_ctrl	Wifi control block
[in]	p_start_ip	Pointer to Start IP address
[in]	p_end_ip	Pointer to End IP address

Return values

SSP_SUCCESS	Successfully started DHCP Server
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_NOT_OPEN	WiFi Driver in not open
SSP_ERR_IN_USE	Device already in use
SSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.

Get the mutex

Start DHCP server

Release the mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStop()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_DhcpServerStop (sf_wifi_qca4010_onchip_stack_ctrl_t *const *p_ctrl*, ULONG * *p_start_ip*, ULONG * *p_end_ip*)

Stops DHCP server.

Implements sf_wifi_qca4010_onchip_stack_api_t::dhcpServerStop Stops DHCP server using on CHIP networking stack support.

Parameters

[in]	p_ctrl	Wifi control block
[in]	p_start_ip	Pointer to Start IP
[in]	p_end_ip	Pointer to End IP

Return values

SSP_SUCCESS	Successfully stopped DHCP Server
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_NOT_OPEN	WiFi Driver is not open
SSP_ERR_IN_USE	Device already in use
SSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.

Get mutex

Stop DHCP server

Release the mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_IPAddressConfigure()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_IPAddressConfigure (sf_wifi_qca4010_onchip_stack_ctrl_t *const *p_ctrl*, sf_wifi_qca4010_onchip_stack_ip_cfg_t *const *p_ip_cfg*)

Configure static IP or enable DHCP.

Implements sf_wifi_qca4010_onchip_stack_api_t::ipAddressConfigure Configures the IP address of the interface or enables DHCP using on CHIP networking stack support.

Parameters

[in]	p_ctrl	Wifi control block
[out]	p_ip_cfg	Pointer to IP Address configuration structure

Return values

SSP_SUCCESS	Successfully configured IP address.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_NOT_OPEN	WiFi Driver in not open
SSP_ERR_UNSUPPORTED	Functionality Unsupported
SSP_ERR_IN_USE	Device already in use
SSP_ERR_WIFI_FAILED	IP configuration failed.
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex

Configure static IP or enable DHCP

Release the mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_Open()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_Open (sf_wifi_qca4010_onchip_stack_ctrl_t * *p_ctrl*, sf_wifi_qca4010_onchip_stack_cfg_t const *const *p_cfg*)

Open the WiFi Device driver to use the Onchip stack support.

Implements sf_wifi_qca4010_onchip_stack_api_t::open Calls the low level WiFi device driver's Open API to Initialize the WiFi Device Driver, for using onchip stack.

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_ALREADY_OPEN	WiFi module is already opened
SSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to initialize
SSP_ERR_INTERNAL	Internal Error occurred

Create Mutex for Synchronization

Get Mutex Lock

Initialize Wifi module

Initialize Socket Interface global variable , if module open is successful or already open

Release Mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_Ping()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_Ping (sf_wifi_qca4010_onchip_stack_ctrl_t *const *p_ctrl*, ULONG * *p_ip_addr*, uint32_t *count*, uint32_t *interval_ms*)

Ping an IP address on the network.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_ip_addr	Pointer to IP address.
[in]	count	Number of ping attempts.
[in]	interval_ms	Interval between ping attempts.

Return values

SSP_SUCCESS	Function completed successfully.
SSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_IN_USE	Device already in use

Get Mutex

Ping specified IP address

Release the mutex

◆ SF_WIFI_QCA4010_ONCHIP_STACK_VersionGet()

ssp_err_t SF_WIFI_QCA4010_ONCHIP_STACK_VersionGet (ssp_version_t *const *p_version*)

This function gets the version information of the underlying driver.

Implements sf_wifi_qca4010_onchip_stack_api_t::versionGet

Parameters

[out]	p_version	Driver version information

Return values

SSP_ERR_ASSERTION	Argument NULL is passed
SSP_SUCCESS	Successfully read version information

5.1.3.51 Socket on QCA4010

Renesas Synergy Software Package Reference » Framework Layer

Implementation of QCA4010 Socket layer over QCA4010 On-Chip stack. More...

Macros

#define	AF_INET (4U) IPV4 Socket Family. More...
#define	AF_INET6 (6U) IPV6 Socket Family.
#define	SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MAJOR (2U)
#define	SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MINOR (0U)

Enumerations

enum	sf_wifi_qca4010_socket_status_t

Functions

ssp_err_t	SF_WIFI_QCA4010_Socket_Open (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, sf_wifi_qca4010_socket_cfg_t const *const p_cfg) Open the WiFi Device driver to use the Socket Layer on WiFi Driver On-Chip stack. More...
ssp_err_t	SF_WIFI_QCA4010_Socket_Close (sf_wifi_qca4010_socket_ctrl_t *const p_ctrl) Close the WiFi Device driver. More...
ssp_err_t	SF_WIFI_QCA4010_Socket_VersionGet (ssp_version_t *const p_version) This function gets the version information of the underlying driver. More...
ssp_err_t	SF_WIFI_QCA4010_Socket_Create (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, sf_wifi_socket_type_t type, uint8_t ipversion) This creates socket for communication. More...

ssp_err_t	SF_WIFI_QCA4010_Socket_Connect (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const struct sockaddr *p_serv_addr, socklen_t addrlen)
ssp_err_t	SF_WIFI_QCA4010_Socket_Disconnect (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no) This closes socket. Close opened socket. More...
ssp_err_t	SF_WIFI_QCA4010_Socket_Send (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, const uint8_t *p_data, uint32_t length, uint32_t timeout_ms)
ssp_err_t	SF_WIFI_QCA4010_Socket_Recv (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint8_t *const p_data, uint32_t length, uint32_t timeout_ms)
ssp_err_t	SF_WIFI_QCA4010_Socket_Status_Get (sf_wifi_qca4010_socket_ctrl_t *p_ctrl, uint8_t socket_no, uint32_t *p_socket_status)

Detailed Description

Implementation of QCA4010 Socket layer over QCA4010 On-Chip stack.

- SF Socket WIFI Framework Interface

Macro Definition Documentation

◆ AF_INET

```
#define AF_INET   (4U)
```

IPV4 Socket Family.

WiFi Interface.

◆ SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MAJOR

```
#define SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MAJOR   (2U)
```

Major Version of code that implements the API defined in this file

◆ SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MINOR

```
#define SF_WIFI_QCA4010_SOCKET_CODE_VERSION_MINOR   (0U)
```

Minor Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sf_wifi_qca4010_socket_status_t

enum sf_wifi_qca4010_socket_status_t
Silex ULPGN Wifi socket status types

Function Documentation

◆ SF_WIFI_QCA4010_Socket_Close()

ssp_err_t SF_WIFI_QCA4010_Socket_Close (sf_wifi_qca4010_socket_ctrl_t *const *p_ctrl*)

Close the WiFi Device driver.

Parameters

[in]	p_ctrl	Socket control block Implements sf_wifi_qca4010_socket_api_t ::close Call the low level WiFi device driver's Close API to close the WiFi Driver.

Return values

SSP_SUCCESS	Suspend the driver functionality.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to close
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Close lower level drivers

Delete thread created for each socket

Release Mutex

Delete the mutex

◆ SF_WIFI_QCA4010_Socket_Connect()

ssp_err_t SF_WIFI_QCA4010_Socket_Connect (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*, const struct sockaddr * *p_serv_addr*, socklen_t *addrlen*)

Connect to a specific IP and Port using socket.

Parameters

[in]	p_ctrl	pointer to Socket Wifi control block
[in]	socket_no	Socket ID number.
[in]	p_serv_addr	Pointer to remote socket address
[in]	addrlen	Size of sock address structure

Return values

SSP_SUCCESS	Socket connection is successful.
SSP_ERR_WIFI_FAILED	Socket connection failed
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Change socket index for multiple socket communication

Connect socket to specified address

For UDP server, specify the destination address and port

Specify the destination address and port for UDP data to be sent

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Create()

ssp_err_t SF_WIFI_QCA4010_Socket_Create (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*, sf_wifi_socket_type_t *type*, uint8_t *ipversion*)

This creates socket for communication.

Parameters

[in]	p_ctrl	pointer to Socket control block
[in]	socket_no	Socket ID number.
[in]	type	Socket type (SOCK_STREAM for TCP and SOCK_DGRAM for UDP)
[in]	ipversion	Socket IP type.(for IPV4 -> AF_INET or for IPV6 -> AF_INET6)

Return values

SSP_SUCCESS	Socket creation is successful
SSP_ERR_WIFI_FAILED	Failed to create socket
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Change socket index for multiple socket communication

Create socket

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Disconnect()

ssp_err_t SF_WIFI_QCA4010_Socket_Disconnect (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*)

This closes socket. Close opened socket.

Parameters

[in]	p_ctrl	pointer to Socket Wifi control block
[in]	socket_no	Socket no

Return values

SSP_SUCCESS	Disconnect and close socket network
SSP_ERR_WIFI_FAILED	Failed to close socket network
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Change socket index for multiple socket communication

If only one UART is used then send escape sequence command

Close the socket network

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Open()

ssp_err_t SF_WIFI_QCA4010_Socket_Open (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, sf_wifi_qca4010_socket_cfg_t const *const *p_cfg*)

Open the WiFi Device driver to use the Socket Layer on WiFi Driver On-Chip stack.

Parameters

[out]	p_ctrl	pointer to Socket control block
[in]	p_cfg	pointer to Socket configuration structure Implements sf_wifi_qca4010_socket_api_t ::open Call the low level WiFi device driver's Open API to Initialize the WiFi Device Driver, for using Socket Layer on On-Chip stack.

Return values

SSP_SUCCESS	Module initialization successful
SSP_ERR_ALREADY_OPEN	WiFi module is already opened
SSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed to initialize
SSP_ERR_INTERNAL	Internal Error occurred
SSP_ERR_INVALID_ARGUMENT	Failed due to invalid argument

Create Mutex for Synchronization

Get Mutex Lock

Socket open

Initialize global variables

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Recv()

ssp_err_t SF_WIFI_QCA4010_Socket_Recv (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*, uint8_t *const *p_data*, uint32_t *length*, uint32_t *timeout_ms*)

Receive data over TCP/UDP from server/client.

Parameters

[in]	p_ctrl	pointer to Socket Wifi control block
[in]	socket_no	Socket ID number.
[out]	p_data	Pointer to data received from socket.
[in]	length	Length of data array used for receive.
[in]	timeout_ms	Timeout to wait for data to be received from socket.

Return values

SSP_SUCCESS	Data reception us successful
SSP_ERR_WIFI_FAILED	Failed to receive data
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Change socket index for multiple socket communication

Receive data which is sent from server/client

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Send()

ssp_err_t SF_WIFI_QCA4010_Socket_Send (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*, const uint8_t * *p_data*, uint32_t *length*, uint32_t *timeout_ms*)

Send data over TCP/UDP to server/client.

Parameters

[in]	p_ctrl	pointer to Socket Wifi control block
[in]	socket_no	Socket ID number.
[in]	p_data	Pointer to data to send.
[in]	length	Length of data to send.
[in]	timeout_ms	Timeout to wait for transmit end event

Return values

SSP_SUCCESS	Data send to client/server is successful.
SSP_ERR_WIFI_FAILED	Failed to send data.
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_IN_USE	Module in use
SSP_ERR_INTERNAL	Internal Error occurred
SSP_ERR_TIMEOUT	Timeout to send data

Get Mutex Lock

Change socket index for multiple socket communication

Send data over data port

Release Mutex

◆ SF_WIFI_QCA4010_Socket_Status_Get()

ssp_err_t SF_WIFI_QCA4010_Socket_Status_Get (sf_wifi_qca4010_socket_ctrl_t * *p_ctrl*, uint8_t *socket_no*, uint32_t * *p_socket_status*)

Get the socket status.

Parameters

[in]	p_ctrl	Socket Wifi control block
[in]	socket_no	Socket ID number.
[out]	p_socket_status	Pointer to an integer to hold the socket status

Return values

SSP_SUCCESS	Socket status obtained successfully.
SSP_ERR_ASSERTION	Assertion error occurred.
SSP_ERR_NOT_OPEN	The instance has not been opened.
SSP_ERR_IN_USE	Module in use
SSP_ERR_WIFI_FAILED	Failed obtain socket status
SSP_ERR_INTERNAL	Internal Error occurred

Get Mutex Lock

Obtain socket status

Release Mutex

◆ SF_WIFI_QCA4010_Socket_VersionGet()

ssp_err_t SF_WIFI_QCA4010_Socket_VersionGet (ssp_version_t *const *p_version*)

This function gets the version information of the underlying driver.

Implements sf_wifi_qca4010_socket_api_t::versionGet

Return values

SSP_SUCCESS	Retrieval of version information is successful
SSP_ERR_ASSERTION	Argument NULL is passed

5.1.3.52 USBX Framework

Renesas Synergy Software Package Reference » Framework Layer

RTOS-integrated USBX adaptation framework for Synergy. Implements USB HOST and DEVICE low level device drivers. More...

Data Structures

struct	UX_DCD_SYNERGY_ED
struct	UX_DCD_SYNERGY_TRANSFER
struct	UX_DCD_SYNERGY_PAYLOAD_TRANSFER
struct	UX_DCD_SYNERGY
struct	UX_HCD_SYNERGY_TRANSFER
struct	UX_HCD_SYNERGY
struct	UX_HCD_SYNERGY_PAYLOAD_TRANSFER
struct	UX_HCD_SYNERGY_FIFO
struct	UX_SYNERGY_ED
struct	UX_SYNERGY_TD
struct	UX_SYNERGY_ISO_TD

Macros

#define	UX_DCD_SYNERGY_SLAVE_CONTROLLER (0x80U)
#define	UX_SYNERGY_DCD_SYSCFG (0x00UL)
#define	UX_SYNERGY_DCD_BUSWAIT_CALC_FREQ_PCLK_CYC (17142857U)
#define	UX_SYNERGY_DCD_MAIN_OSC_24MHz (24000000U)
#define	UX_SYNERGY_DCD_SYSCFG_SCKE (1U<<10)
#define	UX_SYNERGY_DCD_DCP (0)
#define	UX_SYNERGY_DCD_PIPESEL_NO_PIPE (0x000FU)
#define	UX_SYNERGY_DCD_PIPE0_SIZE (256U)
#define	UX_SYNERGY_DCD_COMMAND_STATUS_RESET (0)

#define	UX_SYNERGY_DCD_FIFO_WRITING (2U)
#define	UX_SYNERGY_DCD_FIFO_READING (2U)
#define	UX_SYNERGY_DCD_ED_BRDY (0x00000001U)
#define	UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWR00 (1U<<0)
#define	UX_DCD_SYNERGY_ED_STATUS_UNUSED (0U)
#define	UX_DCD_SYNERGY_ED_STATE_IDLE (0U)
#define	UX_SYNERGY_CONTROLLER (0)
#define	UX_SYNERGY_HC_SYSCFG (0x00UL)
#define	UX_SYNERGY_HC_BUSWAIT_CALC_FREQ_PCLK_CYC (17142857U)
#define	UX_SYNERGY_HC_MAIN_OSC_24MHz (24000000U)
#define	UX_SYNERGY_HC_SYSCFG_SCKE (1U<<10)
#define	UX_SYNERGY_HC_DCP (0)
#define	UX_SYNERGY_HC_PIPESEL_NO_PIPE 0x000f
#define	UX_SYNERGY_HC_PIPE0_SIZE (256)
#define	UX_SYNERGY_HC_AVAILABLE_BANDWIDTH (2304UL)
#define	UX_SYNERGY_HC_COMMAND_STATUS_RESET (0)
#define	UX_SYNERGY_HC_FIFO_WRITING (2)
#define	UX_SYNERGY_HC_FIFO_READING (2)
#define	UX_SYNERGY_HC_ED_BRDY (0x00000001U)
#define	UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWR00 (1U<<0)
#define	UX_SYNERGY_HC_PORT_ENABLED (1)
#define	UX_SYNERGY_ED_STATIC (0x80000000)
#define	UX_SYNERGY_TD_SETUP_PHASE (0x00010000)

Functions

void	usbhs_usb_int_resume_isr (void) This function calls the host interrupt handler or the device interrupt handler.
void	usbfs_int_isr (void) All the USBFS interrupts are handled by this ISR.
void	usbfs_resume_isr (void) VBINT (VBUS interrupt), RESM (Resume interrupt), OVRCR (Over current input), BCHG (Change interrupt), and PDDETINT0 (Bus change interrupt) fire this ISR. This is only used for canceling following standby modes. More...
VOID	ux_dcd_synergy_buffer_empty_interrupt (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, ULONG flag) This function processes the BEMP interrupt for the specific endpoint. More...
VOID	ux_dcd_synergy_buffer_notready_interrupt (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, ULONG flag) This function processes the NRDY(Not Ready) interrupt for the specific endpoint. More...
UINT	ux_dcd_synergy_buffer_read (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function reads from a specified pipe into a buffer. More...
VOID	ux_dcd_synergy_buffer_ready_interrupt (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, ULONG flag) This function enable or disable the BRDY(Ready) interrupt for the pipe. More...
UINT	ux_dcd_synergy_buffer_write (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function writes a data from input buffer to the specified PIPE. More...
VOID	ux_dcd_synergy_current_endpoint_change (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, ULONG direction)

	This function configures the FIFO as per the request specified in endpoint descriptor. More...
ULONG	ux_dcd_synergy_data_buffer_size (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function returns the size of the data buffer and selects the specified pipe. More...
UINT	ux_dcd_synergy_endpoint_create (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_ENDPOINT *endpoint) This function creates a physical endpoint. More...
UINT	ux_dcd_synergy_endpoint_destroy (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_ENDPOINT *endpoint) This function will destroy a physical endpoint. More...
VOID	ux_dcd_synergy_endpoint_nak_set (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function sets a NAK(Not Acknowledged) to an endpoint. More...
UINT	ux_dcd_synergy_endpoint_reset (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_ENDPOINT *endpoint) This function will reset a physical endpoint. More...
UINT	ux_dcd_synergy_remote_wakeup (UX_DCD_SYNERGY *dcd_synergy, ULONG *parameter) This function is called when the device wants to wake up the host. More...
UINT	ux_dcd_synergy_endpoint_stall (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_ENDPOINT *endpoint) This function will stall a physical endpoint. More...
UINT	ux_dcd_synergy_endpoint_status (UX_DCD_SYNERGY *dcd_synergy, ULONG endpoint_index) This function will retrieve the status of the endpoint. More...
ULONG	ux_dcd_synergy_fifo_port_change (UX_DCD_SYNERGY *dcd_synergy,

	UX_DCD_SYNERGY_ED *ed, ULONG direction) This function configures the FIFO port. More...
UINT	ux_dcd_synergy_fifoc_read (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function reads from the hardware FIFO C and stores in the destination buffer. More...
UINT	ux_dcd_synergy_fifo_read (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function reads from the hardware FIFO D0 or D1 and stores in the destination buffer. More...
UINT	ux_dcd_synergy_fifo_read_dma (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, UINT dma_bytes_to_transfer) This function reads the data from HW D0/ D1 FIFO using DMA. More...
UINT	ux_dcd_synergy_fifoc_write (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function writes a data from a buffer into USB FIFO using CPU. More...
VOID	ux_dcd_synergy_fifo_write_software_copy (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_PAYLOAD_TRANSFER *p_payload, VOID *p_fifo, ULONG fifo_sel) USBX DCD FIFO write by software copy. Call a subroutine for selected USB controller hardware. More...
VOID	ux_dcd_synergy_fifo_write_last_bytes (UX_DCD_SYNERGY_PAYLOAD_TRANSFER *p_payload, VOID *p_fifo, ULONG usb_base) USBX DCD FIFO write - Copy last bytes to FIFO by software if the rest bytes are less than FIFO access width. More...
UINT	ux_dcd_synergy_fifod_write (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed) This function writes a buffer to FIFOD0 or FIFOD1. More...
UINT	ux_dcd_synergy_fifod_write_dma (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_ED *ed, UINT dma_bytes_to_transfer)

	This function writes a buffer to FIFOD0 or FIFOD1 using DMA. More...
VOID	ux_dcd_synergy_write_dma_configure (UX_DCD_SYNERGY *dcd_synergy, UX_DCD_SYNERGY_PAYLOAD_TRANSFER *p_payload, ULONG fifo_sel, ULONG endpoint_size) USBX DCD DMA write setup function. Call a subroutine for selected USB controller hardware. More...
VOID	ux_dcd_synergy_fifo_dma_start_write (UX_DCD_SYNERGY *dcd_synergy, UCHAR *p_payload_buffer, VOID *p_fifo_add, VOID *p_fifo_ctrl, VOID *p_fifo_sel) USBX DCD DMA FIFO write - DMA start function. More...
UINT	ux_dcd_synergy_frame_number_get (UX_DCD_SYNERGY *dcd_synergy, ULONG *frame_number) This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes. More...
UINT	ux_dcd_synergy_function (UX_SLAVE_DCD *dcd, UINT function, VOID *parameter) This function act as interface between upper layer USBX device stack and synergy controller. More...
UINT	ux_dcd_synergy_initialize (ULONG dcd_io) This function initializes the USB slave controller for Renesas Synergy MCUs. More...
UINT	ux_dcd_synergy_initialize_transfer_support (ULONG dcd_io, UX_DCD_SYNERGY_TRANSFER *p_transfer_instance) The function initializes the USB slave controller of the Renesas Synergy MCUs with associated DMA transfer modules. More...
UINT	ux_dcd_synergy_initialize_complete (VOID) This function completes the initialization of the USB slave controller for the Renesas Synergy MCUs. More...
VOID	ux_dcd_synergy_interrupt_handler (VOID) This function is the interrupt handler for the synergy controller.

	More...
VOID	ux_dcd_synergy_register_clear (UX_DCD_SYNERGY *dcd_synergy, ULONG synergy_register, USHORT value) This function clears a bit in a register of the synergy. More...
VOID	ux_dcd_synergy_usb_status_register_clear (UX_DCD_SYNERGY *dcd_synergy, ULONG synergy_register, USHORT value) This function clears a bit in a status register of the synergy.To clear the status bits, need to write 0 only to the bits to be cleared. Write 1 to the other bits. More...
ULONG	ux_dcd_synergy_register_read (UX_DCD_SYNERGY *dcd_synergy, ULONG synergy_register) This function reads a synergy USB register. More...
VOID	ux_dcd_synergy_register_set (UX_DCD_SYNERGY *dcd_synergy, ULONG synergy_register, USHORT value) This function set a bit in synergy USB register. More...
VOID	ux_dcd_synergy_register_write (UX_DCD_SYNERGY *dcd_synergy, ULONG synergy_register, USHORT value) This function writes a bit in synergy USB register. More...
UINT	ux_dcd_synergy_transfer_abort (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_TRANSFER *transfer_request) This function will terminate the transfer. More...
UINT	ux_dcd_synergy_transfer_callback (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_TRANSFER *transfer_request, ULONG interrupt_status, ULONG ctsq_mask) This function is invoked under ISR when an event happens on a specific endpoint. More...
UINT	ux_dcd_synergy_transfer_request (UX_DCD_SYNERGY *dcd_synergy, UX_SLAVE_TRANSFER *transfer_request) This function will initiate a transfer to a specific endpoint. If the endpoint is IN, the endpoint register will be set to accept the request. If the endpoint is IN, the endpoint FIFO will be filled with the buffer and the endpoint register set. More...

void	ux_dcd_synergy_disconnect (void) USBX DCD disconnect the USB controller communication from the host.
UINT	ux_dcd_synergy_uninitialize (ULONG dcd_io) USBX DCD un-initialization the USB controller. More...
UINT	ux_dcd_synergy_uninitialize_transfer_support (ULONG dcd_io) The function un-initializes the USB slave controller of the Renesas Synergy MCUs with associated DMA transfer modules. More...
VOID	ux_hcd_synergy_asynch_queue_process (UX_HCD_SYNERGY *hcd_synergy) This function process the asynchronous transactions. The function will identify the USB interrupts occurred associated with an endpoint and will process the interrupts. More...
VOID	ux_hcd_synergy_asynch_queue_process_bemp (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the BEMP(Buffer Empty) interrupt that occurred on a specific ED. More...
VOID	ux_hcd_synergy_asynch_queue_process_brdy (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED. More...
VOID	ux_hcd_synergy_asynch_queue_process_nrdy (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED. More...
VOID	ux_hcd_synergy_asynch_queue_process_sign (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the Setup transaction Error Interrupt. More...
VOID	ux_hcd_synergy_asynch_schedule (UX_HCD_SYNERGY *hcd_synergy) This function schedules new transfers from the control or bulk lists.

	More...
UINT	ux_hcd_synergy_asynchronous_endpoint_create (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will create an asynchronous endpoint. The control and bulk endpoints fall into this category. More...
UINT	ux_hcd_synergy_asynchronous_endpoint_destroy (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will destroy an asynchronous endpoint. The control and bulk endpoints fall into this category. More...
VOID	ux_hcd_synergy_buffer_empty_interrupt (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed, ULONG flag) This function enable or disable the BEMP(Buffer Empty) interrupt for the pipe. More...
VOID	ux_hcd_synergy_buffer_notready_interrupt (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed, ULONG flag) This function enable or disable the NRDY(Not Ready) interrupt for the pipe. More...
UINT	ux_hcd_synergy_buffer_read (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function reads from a specified pipe into a buffer. More...
VOID	ux_hcd_synergy_buffer_ready_interrupt (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed, ULONG flag) This function enable or disable the BRDY(Ready) interrupt for the pipe. More...
UINT	ux_hcd_synergy_buffer_write (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function writes data to the selected FIFO of the endpoint. More...
UINT	ux_hcd_synergy_bulk_endpoint_create (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will create a bulk endpoint. More...

UINT	ux_hcd_synergy_bulk_int_td_add (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function adds a transfer descriptor to an Bulk or INT ED. More...
UINT	ux_hcd_synergy_control_endpoint_create (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will create a control endpoint. More...
UINT	ux_hcd_synergy_control_td_add (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function adds a transfer descriptor to an ED. More...
UINT	ux_hcd_synergy_controller_disable (UX_HCD_SYNERGY *hcd_synergy) This function will disable the Synergy controller. The controller will release all its resources (memory, IO ...). After this, the controller will not send SOF any longer. All transactions should have been completed, all classes should have been closed. More...
VOID	ux_hcd_synergy_current_endpoint_change (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed, ULONG direction) This function change the endpoint in the FIFO. More...
ULONG	ux_hcd_synergy_data_buffer_size (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function returns the size of the buffer data. More...
UX_SYNERGY_ED *	ux_hcd_synergy_ed_obtain (UX_HCD_SYNERGY *hcd_synergy) This function obtains a free ED from the ED list. More...
VOID	ux_hcd_synergy_ed_td_clean (UX_SYNERGY_ED *ed) This function process cleans the ED of all tds except the last dummy TD. More...
VOID	ux_hcd_synergy_endpoint_nak_set (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function sets a NAK(Not Acknowledged) to an endpoint. More...

UINT	ux_hcd_synergy_endpoint_reset (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will reset an endpoint. More...
UINT	ux_hcd_synergy_entry (UX_HCD *hcd, UINT function, VOID *parameter) This function is the entry function to the USB driver from the USB stack. More...
ULONG	ux_hcd_synergy_fifo_port_change (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed, ULONG direction) This function change the port of the FIFO. More...
UINT	ux_hcd_synergy_fifo_read (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function read data from the FIFO configured for the PIPE(FIFO C, D0 or D1). More...
UINT	ux_hcd_synergy_fifoc_write (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function writes a buffer to FIFOC. More...
VOID	ux_hcd_synergy_fifo_write_software_copy (UX_HCD_SYNERGY *hcd_synergy, ULONG payload_length, UCHAR *payload_buffer, VOID *fifo_addr, ULONG fifo_sel) USBX HCD CPU FIFO write by software copy. Call a suitable subroutine for selected USB controller hardware. More...
VOID	ux_hcd_synergy_fifo_write_software_copy_remaining_bytes (UX_HCD_SYNERGY *hcd_synergy, ULONG payload_length, UCHAR *payload_buffer, VOID *fifo_addr) USBX HCD CPU FIFO write - Copy remaining bytes to FIFO by software if the rest bytes are less than FIFO access width. More...
UINT	ux_hcd_synergy_fifod_write (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function writes a buffer data to FIFOD0 or FIFOD1. More...
UINT	ux_hcd_synergy_frame_number_get (UX_HCD_SYNERGY

	*hcd_synergy, ULONG *frame_number) This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes and for timing. More...
VOID	ux_hcd_synergy_frame_number_set (UX_HCD_SYNERGY *hcd_synergy, ULONG frame_number) This function will set the current frame number to the one specified. This function is mostly used for isochronous purpos.es. More...
UINT	ux_hcd_synergy_initialize (UX_HCD *hcd) This function initializes the Synergy controller. More...
UINT	ux_hcd_synergy_initialize_transfer_support (UX_HCD *hcd, const UX_HCD_SYNERGY_TRANSFER *p_transfer_instance) USBX HCD Transfer Support with DMA support. More...
UINT	ux_hcd_synergy_interrupt_endpoint_create (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will create an interrupt endpoint. The interrupt endpoint has an interval of operation from 1 to 255. The Synergy has no hardware scheduler but we still build an interrupt tree similar to the OHCI controller. More...
VOID	ux_hcd_synergy_interrupt_handler (UINT hcd_index) This function is the interrupt handler for the Synergy USB HS controller. Normally an interrupt occurs from the controller when there is either a EOF signal and there has been transfers within the frame or when there is a change on one of the downstream ports. More...
VOID	ux_hcd_synergy_iso_queue_process (UX_HCD_SYNERGY *hcd_synergy) This function process the isochronous transactions that happened in the last frame. More...
VOID	ux_hcd_synergy_iso_queue_process_bemp (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the BEMP(Buffer Empty) Interrupt that occurred on a specific ED used for Isochronous transfer. More...

VOID	ux_hcd_synergy_iso_queue_process_brdy (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED used for isochronous transfer. More...
VOID	ux_hcd_synergy_iso_queue_process_nrdy (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED used for Isochronous transfer. More...
VOID	ux_hcd_synergy_iso_schedule (UX_HCD_SYNERGY *hcd_synergy) This function schedules new transfers from isochronous list. More...
UINT	ux_hcd_synergy_iso_td_add (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function adds a transfer descriptor to an Isochronous Endpoint Descriptor. More...
UINT	ux_hcd_synergy_isochronous_endpoint_create (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function creates an isochronous endpoint. More...
UX_SYNERGY_ISO_TD *	ux_hcd_synergy_isochronous_td_obtain (UX_HCD_SYNERGY *hcd_synergy) This function obtains a free TD from the isochronous TD list. More...
UX_SYNERGY_ED *	ux_hcd_synergy_least_traffic_list_get (UX_HCD_SYNERGY *hcd_synergy) This function return a pointer to the first ED in the periodic tree that has the least traffic registered. More...
UINT	ux_hcd_synergy_periodic_endpoint_destroy (UX_HCD_SYNERGY *hcd_synergy, UX_ENDPOINT *endpoint) This function will destroy an isochronous endpoint. More...
VOID	ux_hcd_synergy_periodic_schedule (UX_HCD_SYNERGY *hcd_synergy) This function schedules new transfers from the periodic interrupt list.

	More...
UINT	ux_hcd_synergy_periodic_tree_create (UX_HCD_SYNERGY *hcd_synergy) This function creates the periodic static tree for the interrupt and isochronous eds. More...
UINT	ux_hcd_synergy_port_disable (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will disable a specific port attached to the root HUB. More...
UINT	ux_hcd_synergy_port_enable (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will enable a specific port attached to the root HUB. More...
UINT	ux_hcd_synergy_port_reset (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will reset a specific port attached to the root HUB. More...
UINT	ux_hcd_synergy_port_resume (UX_HCD_SYNERGY *hcd_synergy, UINT port_index) This function will resume a specific port attached to the root HUB. Present, this function is not supported for resume port. More...
ULONG	ux_hcd_synergy_port_status_get (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will return the status for each port attached to the root HUB. More...
UINT	ux_hcd_synergy_port_suspend (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will suspend a specific port attached to the root HUB. Present, this function is does not supported. More...
UINT	ux_hcd_synergy_power_down_port (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will power down a specific port attached to the root

	HUB. Present, this function is does not supported. More...
UINT	ux_hcd_synergy_power_on_port (UX_HCD_SYNERGY *hcd_synergy, ULONG port_index) This function will power a specific port attached to the root HUB. Present, this function is does not supported. More...
VOID	ux_hcd_synergy_power_root_hubs (UX_HCD_SYNERGY *hcd_synergy) This function will power the root HUB. Present, this function is does not supported. More...
VOID	ux_hcd_synergy_register_clear (UX_HCD_SYNERGY *hcd_synergy, ULONG synergy_register, USHORT value) This function clears flags in a synergy USB register. More...
VOID	ux_hcd_synergy_register_status_clear (UX_HCD_SYNERGY *hcd_synergy, ULONG synergy_register, USHORT value) This function clears a bit in a status register of the synergy controller.To clear the status bits, need to write 0 only to the bits to be cleared. Write 1 to the other bits. More...
ULONG	ux_hcd_synergy_register_read (UX_HCD_SYNERGY *hcd_synergy, ULONG synergy_register) This function reads a data from synergy USB register. More...
VOID	ux_hcd_synergy_register_set (UX_HCD_SYNERGY *hcd_synergy, ULONG synergy_register, USHORT value) This function sets flags in a synergy USB register. More...
VOID	ux_hcd_synergy_register_write (UX_HCD_SYNERGY *hcd_synergy, ULONG synergy_register, USHORT value) This function writes a data to a Synergy USB register. More...
UX_SYNERGY_TD *	ux_hcd_synergy_regular_td_obtain (UX_HCD_SYNERGY *hcd_synergy) This function obtains a free TD from the regular TD list. More...
UINT	ux_hcd_synergy_request_bulk_transfer (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request)

	This function performs a bulk transfer request. A bulk transfer can be larger than the size of the Synergy buffer so it may be required to chain multiple tds to accommodate this transfer request. A bulk transfer is non blocking, so we return before the transfer request is completed. More...
UINT	ux_hcd_synergy_request_control_transfer (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request) This function performs a control transfer from a transfer request. The USB control transfer is in 3 phases (setup, data, status). This function will chain all phases of the control sequence before setting the Synergy endpoint as a candidate for transfer. More...
UINT	ux_hcd_synergy_request_interrupt_transfer (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request) This function performs an interrupt transfer request. An interrupt transfer can only be as large as the MaxpacketField in the endpoint descriptor. This was verified at the USB layer and does not need to be reverified here. More...
UINT	ux_hcd_synergy_request_isochronous_transfer (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request) This function performs an isochronous transfer request. More...
UINT	ux_hcd_synergy_request_transfer (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request) This function is the handler for all the transactions on the USB. The transfer request passed as parameter contains the endpoint and the device descriptors in addition to the type of transaction de be executed. This function routes the transfer request to according to the type of transfer to be executed. More...
UINT	ux_hcd_synergy_td_add (UX_HCD_SYNERGY *hcd_synergy, UX_SYNERGY_ED *ed) This function add new TD for control, Bulk or Interrupt endpoint. More...
UINT	ux_hcd_synergy_transfer_abort (UX_HCD_SYNERGY *hcd_synergy, UX_TRANSFER *transfer_request) This function will abort transactions attached to a transfer request. More...

UINT	ux_hcd_synergy_disable (ULONG ux_hcd_io) This function disables the Synergy HOST controller. More...
UINT	ux_hcd_synergy_uninitialize (ULONG ux_hcd_io) This function un-initializes the Synergy HOST controller. More...
UINT	ux_hcd_synergy_uninitialize_transfer_support (UX_HCD_SYNERGY *hcd_synergy) This function un-initializes the transfer module associated with the USBX HOST controller. More...

Detailed Description

RTOS-integrated USBX adaptation framework for Synergy. Implements USB HOST and DEVICE low level device drivers.

USBX Component SYNERGY Controller Driver

Macro Definition Documentation

◆ UX_DCD_SYNERGY_ED_STATE_IDLE

```
#define UX_DCD_SYNERGY_ED_STATE_IDLE   (0U)
```

Define USB SYNERGY physical endpoint state machine definition.

◆ UX_DCD_SYNERGY_ED_STATUS_UNUSED

```
#define UX_DCD_SYNERGY_ED_STATUS_UNUSED   (0U)
```

Define USB SYNERGY physical endpoint status definition.

◆ UX_DCD_SYNERGY_SLAVE_CONTROLLER

```
#define UX_DCD_SYNERGY_SLAVE_CONTROLLER   (0x80U)
```

Define SYNERGY generic equivalences.

◆ UX_SYNERGY_CONTROLLER

```
#define UX_SYNERGY_CONTROLLER   (0)
```

Define Synergy generic definitions.

◆ UX_SYNERGY_DCD_BUSWAIT_CALC_FREQ_PCLK_CYC

#define UX_SYNERGY_DCD_BUSWAIT_CALC_FREQ_PCLK_CYC (17142857U)
Register access wait cycles for USBHS controller. 7cycles at Peripheral clock 120MHz

◆ UX_SYNERGY_DCD_COMMAND_STATUS_RESET

#define UX_SYNERGY_DCD_COMMAND_STATUS_RESET (0)
Define synergy initialization values.

◆ UX_SYNERGY_DCD_DCP

#define UX_SYNERGY_DCD_DCP (0)
Define synergy command/status bitmaps.

◆ UX_SYNERGY_DCD_ED_BRDY

#define UX_SYNERGY_DCD_ED_BRDY (0x00000001U)
Define synergy physical endpoint definitions.

◆ UX_SYNERGY_DCD_FIFO_READING

#define UX_SYNERGY_DCD_FIFO_READING (2U)
Define synergy FIFO read completion code.

◆ UX_SYNERGY_DCD_FIFO_WRITING

#define UX_SYNERGY_DCD_FIFO_WRITING (2U)
Define synergy FIFO write completion code.

◆ UX_SYNERGY_DCD_MAIN_OSC_24MHz

#define UX_SYNERGY_DCD_MAIN_OSC_24MHz (24000000U)
Supported USBMCLK frequency for S7G2 and S5D9.

◆ UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWR00

#define UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWR00 (1U<<0)
PHY Cross Point Adjustment, note that Hardware Manual to be updated(0xE->0x5)

◆ UX_SYNERGY_DCD_PIPE0_SIZE

#define UX_SYNERGY_DCD_PIPE0_SIZE (256U)
Define synergy fifo definition.

◆ UX_SYNERGY_DCD_PIPESEL_NO_PIPE

#define UX_SYNERGY_DCD_PIPESEL_NO_PIPE (0x000FU)
Define synergy PIPE selection definitions.

◆ UX_SYNERGY_DCD_SYSCFG

#define UX_SYNERGY_DCD_SYSCFG (0x00UL)
Define SYNERGY HCOR register mapping.

◆ UX_SYNERGY_DCD_SYSCFG_SCKE

#define UX_SYNERGY_DCD_SYSCFG_SCKE (1U<<10)
Define SYNERGY control register values.

◆ UX_SYNERGY_ED_STATIC

#define UX_SYNERGY_ED_STATIC (0x80000000)
Define Synergy ED bitmap.

◆ UX_SYNERGY_HC_AVAILABLE_BANDWIDTH

#define UX_SYNERGY_HC_AVAILABLE_BANDWIDTH (2304UL)
Define Synergy static definition. This macro is used for checking the available bandwidth for periodic transfers(Isochronous and Interrupt) Maximum bandwidth is calculated as {2048byes(2x ISO PIPEs) + 256bytes(4x INT PIPEs)} for high-speed operation.

◆ UX_SYNERGY_HC_BUSWAIT_CALC_FREQ_PCLK_CYC

#define UX_SYNERGY_HC_BUSWAIT_CALC_FREQ_PCLK_CYC (17142857U)
Register access wait cycles for USBHS controller. 7cycles at Peripheral clock 120MHz

◆ UX_SYNERGY_HC_COMMAND_STATUS_RESET

#define UX_SYNERGY_HC_COMMAND_STATUS_RESET (0)
Define Synergy initialization values.

◆ UX_SYNERGY_HC_DCP

#define UX_SYNERGY_HC_DCP (0)
Define Synergy HCOR command/status bitmaps.

◆ UX_SYNERGY_HC_ED_BRDY

#define UX_SYNERGY_HC_ED_BRDY (0x00000001U)
Define Synergy physical endpoint definitions.

◆ UX_SYNERGY_HC_FIFO_READING

#define UX_SYNERGY_HC_FIFO_READING (2)
Define Synergy FIFO read completion code.

◆ UX_SYNERGY_HC_FIFO_WRITING

#define UX_SYNERGY_HC_FIFO_WRITING (2)
Define Synergy FIFO write completion code.

◆ UX_SYNERGY_HC_MAIN_OSC_24MHz

#define UX_SYNERGY_HC_MAIN_OSC_24MHz (24000000U)
Supported USBMCLK frequency for S7G2 and S5D9.

◆ UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWR00

#define UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWR00 (1U<<0)
PHY Cross Point Adjustment, note that Hardware Manual to be updated(0xE->0x5)

◆ UX_SYNERGY_HC_PIPE0_SIZE

#define UX_SYNERGY_HC_PIPE0_SIZE (256)
Define Synergy fifo definition.

◆ UX_SYNERGY_HC_PIPESEL_NO_PIPE

#define UX_SYNERGY_HC_PIPESEL_NO_PIPE 0x000f
Define Synergy PIPE selection definitions.

◆ UX_SYNERGY_HC_PORT_ENABLED

#define UX_SYNERGY_HC_PORT_ENABLED (1)
Define Synergy Root hub states.

◆ UX_SYNERGY_HC_SYSCFG

#define UX_SYNERGY_HC_SYSCFG (0x00UL)
Protection against no definition of Synergy controller.

◆ UX_SYNERGY_HC_SYSCFG_SCKE

#define UX_SYNERGY_HC_SYSCFG_SCKE (1U<<10)
Define Synergy control register values.

◆ UX_SYNERGY_TD_SETUP_PHASE

#define UX_SYNERGY_TD_SETUP_PHASE (0x00010000)
Define Synergy TD bitmap.

Function Documentation

◆ usbfs_resume_isr()

void usbfs_resume_isr (void)
VBINT (VBUS interrupt), RESM (Resume interrupt), OVRCR (Over current input), BCHG (Change interrupt), and PDDETINT0 (Bus change interrupt) fire this ISR. This is only used for canceling following standby modes. • Canceling the software standby mode • Canceling deep standby mode

◆ ux_dcd_synergy_buffer_empty_interrupt()

VOID ux_dcd_synergy_buffer_empty_interrupt (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, ULONG *flag*)
This function processes the BEMP interrupt for the specific endpoint. **Parameters**

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to physical Endpoint(ED) control block
[in]	flag	Flag to enable or disable the buffer empty interrupt.

◆ ux_dcd_synergy_buffer_notready_interrupt()

VOID ux_dcd_synergy_buffer_notready_interrupt (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, ULONG *flag*)
This function processes the NRDY(Not Ready) interrupt for the specific endpoint. **Parameters**

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to physical Endpoint(ED) control block
[in]	flag	Flag for DCD synergy enable or disable.

◆ ux_dcd_synergy_buffer_read()

UINT ux_dcd_synergy_buffer_read (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function reads from a specified pipe into a buffer.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block

Return values

UX_SUCCESS	Read a data from buffer successfully.
UX_ERROR	Unable to read a data from buffer.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_fifo_read()

◆ ux_dcd_synergy_buffer_ready_interrupt()

VOID ux_dcd_synergy_buffer_ready_interrupt (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, ULONG *flag*)

This function enable or disable the BRDY(Ready) interrupt for the pipe.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block
[in]	flag	Check whether DCD synergy is enable or disable.

◆ ux_dcd_synergy_buffer_write()

UINT ux_dcd_synergy_buffer_write (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function writes a data from input buffer to the specified PIPE.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block

Return values

UX_SUCCESS	Write a data to FIFO(D0, D1 and C) successfully.
UX_ERROR	Unable to write a data to FIFO(D0, D1 and C).

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_fifod_write()
- ux_dcd_synergy_fifoc_write()

◆ ux_dcd_synergy_current_endpoint_change()

VOID ux_dcd_synergy_current_endpoint_change (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, ULONG *direction*)

This function configures the FIFO as per the request specified in endpoint descriptor.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block
[in]	direction	Endpoint direction

◆ ux_dcd_synergy_data_buffer_size()

ULONG ux_dcd_synergy_data_buffer_size (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function returns the size of the data buffer and selects the specified pipe.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block

Return values

buffer_size	Maximum packet size.

◆ ux_dcd_synergy_endpoint_create()

UINT ux_dcd_synergy_endpoint_create (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_ENDPOINT * *endpoint*)

This function creates a physical endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	endpoint	Pointer to a Device Controller Endpoint structure.

Return values

UX_SUCCESS	Endpoint is created successfully.
UX_ERROR	Buffer is not free or endpoint creation is unsuccessful.
UX_NO_ED_AVAILABLE	Endpoint is already in use.

◆ ux_dcd_synergy_endpoint_destroy()

UINT ux_dcd_synergy_endpoint_destroy (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_ENDPOINT * *endpoint*)

This function will destroy a physical endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	endpoint	Pointer to a Device Controller Endpoint structure.

Return values

UX_SUCCESS	Endpoint is destroyed successfully.

◆ ux_dcd_synergy_endpoint_nak_set()

VOID ux_dcd_synergy_endpoint_nak_set (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function sets a NAK(Not Acknowledged) to an endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block

◆ ux_dcd_synergy_endpoint_reset()

UINT ux_dcd_synergy_endpoint_reset (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_ENDPOINT * *endpoint*)

This function will reset a physical endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	endpoint	Pointer to a Device Controller Endpoint structure.

Return values

UX_SUCCESS	Endpoint is reset successfully.
UX_NO_ED_AVAILABLE	Device Controller Endpoint structure pointer is NULL

Abort the transfer request on this endpoint.

◆ ux_dcd_synergy_endpoint_stall()

UINT ux_dcd_synergy_endpoint_stall (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_ENDPOINT * *endpoint*)

This function will stall a physical endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	endpoint	Pointer to a Device Controller Endpoint structure.

Return values

UX_SUCCESS	Endpoint is stalled successfully.
UX_NO_ED_AVAILABLE	Device Controller Endpoint control pointer is Null.

◆ ux_dcd_synergy_endpoint_status()

UINT ux_dcd_synergy_endpoint_status (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *endpoint_index*)

This function will retrieve the status of the endpoint.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	endpoint_index	Endpoint number who's status is to be known.

Return values

UX_ERROR	Endpoint already in use.
UX_FALSE	Endpoint is stalled.
UX_TRUE	Endpoint is not stalled.

◆ ux_dcd_synergy_fifo_dma_start_write()

VOID ux_dcd_synergy_fifo_dma_start_write (UX_DCD_SYNERGY * *dcd_synergy*, UCHAR * *p_payload_buffer*, VOID * *p_fifo_add*, VOID * *p_fifo_ctrl*, VOID * *p_fifo_sel*)

USBX DCD DMA FIFO write - DMA start function.

Parameters

[in]	dcd_synergy	Pointer to the DCD control block
[in,out]	p_payload_buffer	Pointer to a payload buffer
[in]	p_fifo_add	FIFO register address
[in]	p_fifo_ctrl	FIFO port control register address
[in]	p_fifo_sel	FIFO port selection register address

◆ ux_dcd_synergy_fifo_port_change()

ULONG ux_dcd_synergy_fifo_port_change (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, ULONG *direction*)

This function configures the FIFO port.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	ed	Pointer to a physical Endpoint(ED) control block
[in]	direction	Direction to switch

Return values

UX_ERROR	Unable to change fifo port.

Returns

synergy_register Current endpoint index(pipe)

◆ ux_dcd_synergy_fifo_read()

UINT ux_dcd_synergy_fifo_read (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function reads from the hardware FIFO D0 or D1 and stores in the destination buffer.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_READ_OVER	FIFO read overflow.
UX_SYNERGY_DCD_FIFO_READ_SHORT	Short packet is received.
UX_SYNERGY_DCD_FIFO_READING	Continue reading FIFO.

◆ ux_dcd_synergy_fifo_read_dma()

UINT ux_dcd_synergy_fifo_read_dma (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, UINT *dma_bytes_to_transfer*)

This function reads the data from HW D0/ D1 FIFO using DMA.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block
[in,out]	dma_bytes_to_transfer	: No of bytes to be transferred using DMA

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_READ_OVER	FIFO read overflow.
UX_SYNERGY_DCD_FIFO_READ_SHORT	Short packet is received.
UX_SYNERGY_DCD_FIFO_READING	Continue reading FIFO.

◆ ux_dcd_synergy_fifo_write_last_bytes()

VOID ux_dcd_synergy_fifo_write_last_bytes (UX_DCD_SYNERGY_PAYLOAD_TRANSFER * *p_payload*, VOID * *p_fifo*, ULONG *usb_base*)

USBX DCD FIFO write - Copy last bytes to FIFO by software if the rest bytes are less than FIFO access width.

Parameters

[in,out]	p_payload	Pointer to a payload transfer structure
[in]	p_fifo	FIFO register address
[in]	usb_base	USB controller hardware base address

◆ ux_dcd_synergy_fifo_write_software_copy()

VOID ux_dcd_synergy_fifo_write_software_copy (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_PAYLOAD_TRANSFER * *p_payload*, VOID * *p_fifo*, ULONG *fifo_sel*)

USBX DCD FIFO write by software copy. Call a subroutine for selected USB controller hardware.

Parameters

[in]	dcd_synergy	Pointer to the DCD control block
[in,out]	p_payload	Pointer to a payload transfer structure
[in]	p_fifo	FIFO register address
[in]	fifo_sel	FIFO select register

◆ ux_dcd_synergy_fifoc_read()

UINT ux_dcd_synergy_fifoc_read (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function reads from the hardware FIFO C and stores in the destination buffer.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_READ_OVER	FIFO read overflow.
UX_SYNERGY_DCD_FIFO_READ_SHORT	Short packet is received.
UX_SYNERGY_DCD_FIFO_READING	Continue reading FIFO.

◆ ux_dcd_synergy_fifoc_write()

UINT ux_dcd_synergy_fifoc_write (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function writes a data from a buffer into USB FIFO using CPU.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_WRITE_END	Status for fifo write ends.
UX_SYNERGY_DCD_FIFO_WRITING	Status for fifo multiple writes.

◆ ux_dcd_synergy_fifod_write()

UINT ux_dcd_synergy_fifod_write (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*)

This function writes a buffer to FIFOD0 or FIFOD1.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_WRITE_END	Write ends of FIFO.
UX_SYNERGY_DCD_FIFO_WRITING	Continue multiple write to FIFO.

◆ ux_dcd_synergy_fifod_write_dma()

UINT ux_dcd_synergy_fifod_write_dma (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_ED * *ed*, UINT *dma_bytes_to_transfer*)

This function writes a buffer to FIFOD0 or FIFOD1 using DMA.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	ed	: Pointer to a physical Endpoint(ED) control block
[in,out]	dma_bytes_to_transfer	: No of bytes to be transferred using DMA

Return values

UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_WRITE_END	Write ends of FIFO.
UX_SYNERGY_DCD_FIFO_WRITING	Continue multiple write to FIFO.
UX_SYNERGY_DCD_FIFO_WRITE_ERROR	Return error if timeout occurs or endpoint reset occurs.

Setup DMA transfer.

Start DMA transfer by software control.

Wait till DMA transfer is done.

Return error, if semaphore timeouts or endpoint reset occurs.

Wait for certain time - if the buffer is not ready return error

Clear the DELSR.n.IR flag by starting the dummy DMA transfer.

◆ ux_dcd_synergy_frame_number_get()

UINT ux_dcd_synergy_frame_number_get (UX_DCD_SYNERGY * *dcd_synergy*, ULONG * *frame_number*)

This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	frame_number	: Pointer to a frame number in use

Return values

UX_SUCCESS	In use frame number is returned successfully.

◆ ux_dcd_synergy_function()

UINT ux_dcd_synergy_function (UX_SLAVE_DCD * *dcd*, UINT *function*, VOID * *parameter*)

This function act as interface between upper layer USBX device stack and synergy controller.

Parameters

[in,out]	dcd	: Pointer to a USBX control block.
[in,out]	function	: Function requested to be despatched.
[in,out]	parameter	: Pointer to requested function parameters.

Return values

UX_CONTROLLER_UNKNOWN	Desired controller is not specified.
UX_FUNCTION_NOT_SUPPORTED	DCD function is not supported by the controller.
UX_SUCCESS	DCD function despatched successfully.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_frame_number_get()
- ux_dcd_synergy_transfer_request()
- ux_dcd_synergy_transfer_abort()
- ux_dcd_synergy_endpoint_create()
- ux_dcd_synergy_endpoint_destroy()
- ux_dcd_synergy_endpoint_reset()
- ux_dcd_synergy_endpoint_stall()
- ux_dcd_synergy_endpoint_status()

◆ ux_dcd_synergy_initialize()

UINT ux_dcd_synergy_initialize (ULONG *dcd_io*)

This function initializes the USB slave controller for Renesas Synergy MCUs.

Parameters

[in,out]	dcd_io	Address of DCD

Returns

See Common Error Codes for other possible return codes or causes.

```
This function calls:
* ux_dcd_synergy_initialize_common()
```

◆ ux_dcd_synergy_initialize_complete()

UINT ux_dcd_synergy_initialize_complete (VOID)
This function completes the initialization of the USB slave controller for the Renesas Synergy MCUs. **Return values** UX_SUCCESS — USB slave is initialized successfully.

◆ ux_dcd_synergy_initialize_transfer_support()

UINT ux_dcd_synergy_initialize_transfer_support (ULONG *dcd_io*, UX_DCD_SYNERGY_TRANSFER * *p_transfer_instance*)

The function initializes the USB slave controller of the Renesas Synergy MCUs with associated DMA transfer modules.

Parameters

[in]	dcd_io	Address of the USB controller.
[in]	p_transfer_instance	Pointer to Synergy Transfer module instances.

Return values

UX_SUCCESS	Completed the USB controller initialization successfully.
UX_CONTROLLER_INIT_FAILED	Failed to initialize the USB controller.
UX_MEMORY_INSUFFICIENT	Memory was not allocated properly for the Synergy DCD instance.

Get the pointer to the Synergy DCD instance.

To begin, initialize D0 and D1 FIFO as free.

Initialize Transfer instances.

Setup the Transfer module for transmission.

DTC is not supported in S1 series - so no need of activation source

Clear the DMA transfer end callback flag.

Open Transfer module for transmission.

Setup the Transfer module for reception.

Note address mode in rx is reverst of tx: Destination (which is ram) is incremented and source (which is FIFO_1 buffer)

DTC is not supported in S1 series - so no need of activation source

Clear the DMA transfer end callback flag.

Open Transfer module for reception.

Return successful completion.

◆ ux_dcd_synergy_interrupt_handler()

VOID ux_dcd_synergy_interrupt_handler (VOID)

This function is the interrupt handler for the synergy controller.

The controller will trigger an interrupt when something happens on an endpoint whose mask has been set in the interrupt enable register.

Get the pointer to the DCD.

Get the pointer to the synergy DCD.

Get the pointer to the device.

Read the interrupt status register from the controller.

Check if we have an RESUME.

Check the source of the interrupt. Is it VBUS transition?

Check the source of the interrupt. Is it Device State transition (DVST) ?

We enter this state when there is a Bus Reset.

If the device is marked as configured, the device is reset.

Decide what speed is used by the host, read DVSTCTR and isolate speed.

Check if we have a BEMP interrupt.

Check if we have a BRDY interrupt.

Check if we have a NRDY interrupt.

Check if we have a SETUP transaction phase.

◆ ux_dcd_synergy_register_clear()

VOID ux_dcd_synergy_register_clear (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function clears a bit in a register of the synergy.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to clear
[in,out]	value	: Value to clear

◆ ux_dcd_synergy_register_read()

ULONG ux_dcd_synergy_register_read (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *synergy_register*)

This function reads a synergy USB register.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to read

Return values

dcd_reg	Value read from USB register.

◆ ux_dcd_synergy_register_set()

VOID ux_dcd_synergy_register_set (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function set a bit in synergy USB register.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to set
[in,out]	value	: Value to set

◆ ux_dcd_synergy_register_write()

VOID ux_dcd_synergy_register_write (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function writes a bit in synergy USB register.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to write
[in,out]	value	: Value to write

◆ ux_dcd_synergy_remote_wakeup()

UINT ux_dcd_synergy_remote_wakeup (UX_DCD_SYNERGY * *dcd_synergy*, ULONG * *parameter*)

This function is called when the device wants to wake up the host.

Parameters

[in]	dcd_synergy	Pointer to a DCD control block
[in]	parameter	Pointer to a remote wakeup parameter.

Return values

UX_SUCCESS	Remote wakeup signaled to the USB HOST successfully.
UX_ERROR	Remote wakeup signal failed from DEVICE to the USB HOST.

◆ ux_dcd_synergy_transfer_abort()

UINT ux_dcd_synergy_transfer_abort (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_TRANSFER * *transfer_request*)

This function will terminate the transfer.

Parameters

[in]	dcd_synergy	: Pointer to a DCD control block
[in]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Transfer Abort is initiated successfully.

Set the ACLRM bit to 1 and then to 0 for clearing FIFO buffers.

Clear the FIFO buffer memory.

◆ ux_dcd_synergy_transfer_callback()

UINT ux_dcd_synergy_transfer_callback (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_TRANSFER * *transfer_request*, ULONG *interrupt_status*, ULONG *ctsq_mask*)

This function is invoked under ISR when an event happens on a specific endpoint.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	transfer_request	: Pointer to USBX Device Transfer Request structure
[in,out]	interrupt_status	: Check if we have SETUP condition or BRDY or BEMP interrupt.
[in,out]	ctsq_mask	: Mask to isolate the CTSQ field.

Return values

UX_SUCCESS	Function is invoked under ISR successfully.

◆ ux_dcd_synergy_transfer_request()

UINT ux_dcd_synergy_transfer_request (UX_DCD_SYNERGY * *dcd_synergy*, UX_SLAVE_TRANSFER * *transfer_request*)

This function will initiate a transfer to a specific endpoint. If the endpoint is IN, the endpoint register will be set to accept the request. If the endpoint is IN, the endpoint FIFO will be filled with the buffer and the endpoint register set.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Transfer to a specific endpoint is initiated successfully.
ux_slave_transfer_request_completion_code	Pointer to structure UX_SLAVE_TRANSFER(transfer request completion code).
UX_TRANSFER_ERROR	Transfer is completed with error.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_buffer_write()

Set the ACLRM bit to 1 and then to 0 for clearing FIFO buffers.

Clear the D1 FIFO buffer memory.

Clear the CFIFO buffer memory.

Clean the pending semaphore due to timeout on this transfer request.

Check the completion code, and if it is not successful abort this transfer and return the error to the caller

Check the completion code, and if it is not successful abort this transfer and return the error to the caller

◆ ux_dcd_synergy_uninitialize()

UINT ux_dcd_synergy_uninitialize (ULONG *dcd_io*)

USBX DCD un-initialization the USB controller.

Parameters

[in]	dcd_io	Address of the USB controller.

Return values

UX_SUCCESS	Completed the USB controller Un-initialization successfully.
UX_DCD_SYNERGY_UNINIT_FAILED	Failed to Un-initialize the USB controller.

Disable interrupt requests

uninitialize and disable DMA support

Stop the clock to the USB module. The SCKE clearing is required for USBFS controller but not for USBHS

Reset USB Module.

Clear the Pending IRQ in NVIC

Disable the IRQ in NVIC

Stop the module usage

Free up resource.

◆ ux_dcd_synergy_uninitialize_transfer_support()

UINT ux_dcd_synergy_uninitialize_transfer_support (ULONG *dcd_io*)

The function un-initializes the USB slave controller of the Renesas Synergy MCUs with associated DMA transfer modules.

Parameters

[in]	dcd_io	Address of the USB controller.

Return values

UX_SUCCESS	Completed the USB controller Un-initialization successfully.
UX_DCD_SYNERGY_UNINIT_FAILED	Failed to Un-initialize the USB controller.

◆ ux_dcd_synergy_usb_status_register_clear()

VOID ux_dcd_synergy_usb_status_register_clear (UX_DCD_SYNERGY * *dcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function clears a bit in a status register of the synergy.To clear the status bits, need to write 0 only to the bits to be cleared. Write 1 to the other bits.

Parameters

[in,out]	dcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to clear
[in,out]	value	: Value to clear

◆ ux_dcd_synergy_write_dma_configure()

VOID ux_dcd_synergy_write_dma_configure (UX_DCD_SYNERGY * *dcd_synergy*, UX_DCD_SYNERGY_PAYLOAD_TRANSFER * *p_payload*, ULONG *fifo_sel*, ULONG *endpoint_size*)

USBX DCD DMA write setup function. Call a subroutine for selected USB controller hardware.

Parameters

[in]	dcd_synergy	Pointer to the DCD control block
[in,out]	p_payload	Pointer to a payload transfer structure
[in]	fifo_sel	FIFO select register
[in]	endpoint_size	Endpoint size

◆ ux_hcd_synergy_asynch_queue_process()

VOID ux_hcd_synergy_asynch_queue_process (UX_HCD_SYNERGY * *hcd_synergy*)

This function process the asynchronous transactions. The function will identify the USB interrupts occurred associated with an endpoint and will process the interrupts.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_asynch_queue_process_bemp()

VOID ux_hcd_synergy_asynch_queue_process_bemp (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the BEMP(Buffer Empty) interrupt that occurred on a specific ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_asynch_queue_process_brdy()

VOID ux_hcd_synergy_asynch_queue_process_brdy (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_asynch_queue_process_nrdy()

VOID ux_hcd_synergy_asynch_queue_process_nrdy (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- UX_TRANSFER::ux_transfer_request_completion_function

◆ ux_hcd_synergy_asynch_queue_process_sign()

VOID ux_hcd_synergy_asynch_queue_process_sign (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the Setup transaction Error Interrupt.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_asynch_schedule()

VOID ux_hcd_synergy_asynch_schedule (UX_HCD_SYNERGY * *hcd_synergy*)

This function schedules new transfers from the control or bulk lists.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_asynchronous_endpoint_create()

UINT ux_hcd_synergy_asynchronous_endpoint_create (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will create an asynchronous endpoint. The control and bulk endpoints fall into this category.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to endpoint

Return values

UX_NO_ED_AVAILABLE	ED for new endpoint not available.
UX_NO_TD_AVAILABLE	Dummy TD not available for terminating the ED transfer chain.
UX_SUCCESS	Asynchronous endpoint created successfully.

◆ ux_hcd_synergy_asynchronous_endpoint_destroy()

UINT ux_hcd_synergy_asynchronous_endpoint_destroy (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will destroy an asynchronous endpoint. The control and bulk endpoints fall into this category.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to endpoint

Return values

UX_ENDPOINT_HANDLE_UNKNOWN	Physical endpoint has not been initialized properly.
UX_SUCCESS	Asynchronous endpoint destroyed successfully.

◆ ux_hcd_synergy_buffer_empty_interrupt()

VOID ux_hcd_synergy_buffer_empty_interrupt (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*, ULONG *flag*)

This function enable or disable the BEMP(Buffer Empty) interrupt for the pipe.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to physical Endpoint(ED) control block
[in]	flag	: Check whether DCD synergy is enable or disable.

Reset the BEMPE, NRDYE, BRDYE bits.

◆ ux_hcd_synergy_buffer_notready_interrupt()

VOID ux_hcd_synergy_buffer_notready_interrupt (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*, ULONG *flag*)

This function enable or disable the NRDY(Not Ready) interrupt for the pipe.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to physical Endpoint(ED) control block
[in]	flag	: Check whether DCD synergy is enable or disable.

◆ ux_hcd_synergy_buffer_read()

UINT ux_hcd_synergy_buffer_read (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function reads from a specified pipe into a buffer.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to a physical Endpoint(ED) control block

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_fifo_read()

◆ ux_hcd_synergy_buffer_ready_interrupt()

VOID ux_hcd_synergy_buffer_ready_interrupt (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*, ULONG *flag*)

This function enable or disable the BRDY(Ready) interrupt for the pipe.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to physical Endpoint(ED) control block
[in]	flag	: Check whether DCD synergy is enable or disable.

◆ ux_hcd_synergy_buffer_write()

UINT ux_hcd_synergy_buffer_write (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function writes data to the selected FIFO of the endpoint.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to a physical Endpoint(ED) control block

Return values

UX_SUCCESS	Buffer written to the specified PIPE successfully.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_fifod_write()
- ux_hcd_synergy_fifoc_write()

◆ ux_hcd_synergy_bulk_endpoint_create()

UINT ux_hcd_synergy_bulk_endpoint_create (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will create a bulk endpoint.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to a USBX Endpoint Container structure

Return values

UX_ERROR	PIPE is not available for bulk endpoint creation .
UX_SUCCESS	Bulk endpoints created successfully.
UX_NO_ED_AVAILABLE	ED for bulk endpoint is not available.
UX_NO_TD_AVAILABLE	Dummy TD for terminating the ED transfer chain is not available.

Limit the max packet size to the size the HW supports.

◆ ux_hcd_synergy_bulk_int_td_add()

UINT ux_hcd_synergy_bulk_int_td_add (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function adds a transfer descriptor to an Bulk or INT ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to a Synergy ED structure

Return values

UX_SUCCESS	Transfer descriptor added successfully.

Clear the BRDY and BEMP status for this pipe.

◆ ux_hcd_synergy_control_endpoint_create()

UINT ux_hcd_synergy_control_endpoint_create (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will create a control endpoint.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	Control endpoint created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED for control endpoint.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for control endpoint.

◆ ux_hcd_synergy_control_td_add()

UINT ux_hcd_synergy_control_td_add (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function adds a transfer descriptor to an ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to Synergy ED structure

Return values

UX_SUCCESS	Transfer descriptor added to an ED successfully.

Get transmit descriptors.

Set TD into response pending state.

Check data, status for setup phase.

We are processing a SETUP phase. Set the device address register if different.

And store it. Note that the device address is not an index.

DEVADDm.USBSPD[1:0] is set by the Software after the speed of the device is obtained and reset after connecting/disconnecting the device every time. To set the TRNENSEL bit, when LS device is connected to FS Hub, we are indirectly examining the speed of the device by examining DEVADDm.USBSPD[1:0].

Check if the speed of the connected HUB is FS

Clear the address field first.

Store the new address but leave the MPS field intact.

Set the buffer address to be accessed.

Copy the payload of the control transfer into each register : Request.

Copy the payload of the control transfer into each register : Value.

Copy the payload of the control transfer into each register : Index.

Copy the payload of the control transfer into each register : Length.

Start transmission.

We are processing data/status stage of control transfer. Check direction now.

This is an IN. Reset the PID mask register.

Set PIPE0 FIFO in in status.

Set DATA0-DATA1 toggle.

We are doing a read. Reset the Direction bit in the DCPCFG register.

Clear the FIFO buffer memory.

Enable the Buffer empty interrupt.

Enable the Ready interrupt.

Start transmission - set PID to NAK then set PID to BUF.

Must be an OUT now.

Clear the FIFO buffer memory.

Set PID to NAK.

Set DATA0-DATA1 toggle.

Write the buffer to the pipe.

Check status.

Return successful completion.

◆ ux_hcd_synergy_controller_disable()

UINT ux_hcd_synergy_controller_disable (UX_HCD_SYNERGY * *hcd_synergy*)

This function will disable the Synergy controller. The controller will release all its resources (memory, IO ...). After this, the controller will not send SOF any longer. All transactions should have been completed, all classes should have been closed.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

UX_SUCCESS	Synergy controller disabled successfully.

◆ ux_hcd_synergy_current_endpoint_change()

VOID ux_hcd_synergy_current_endpoint_change (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*, ULONG *direction*)

This function change the endpoint in the FIFO.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure
[in]	direction	: Direction to transfer

◆ ux_hcd_synergy_data_buffer_size()

ULONG ux_hcd_synergy_data_buffer_size (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function returns the size of the buffer data.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure

Return values

buffer_size	buffer size

◆ ux_hcd_synergy_disable()

UINT ux_hcd_synergy_disable (ULONG *ux_hcd_io*)

This function disables the Synergy HOST controller.

Parameters

[in]	ux_hcd_io	: HCD controller base address

Return values

UX_SUCCESS	HCD controller disabled successfully.
UX_SYNERGY_UNINIT_FAILED	HCD controller un-initialization failed.

◆ ux_hcd_synergy_ed_obtain()

UX_SYNERGY_ED* ux_hcd_synergy_ed_obtain (UX_HCD_SYNERGY * *hcd_synergy*)

This function obtains a free ED from the ED list.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

UX_NULL	No available ED in the ED list.

Returns

ed Endpoint descriptor pointer address.

◆ ux_hcd_synergy_ed_td_clean()

VOID ux_hcd_synergy_ed_td_clean (UX_SYNERGY_ED * *ed*)

This function process cleans the ED of all tds except the last dummy TD.

Parameters

[in,out]	ed	: Pointer to Synergy ED structure

◆ ux_hcd_synergy_endpoint_nak_set()

VOID ux_hcd_synergy_endpoint_nak_set (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function sets a NAK(Not Acknowledged) to an endpoint.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure

◆ ux_hcd_synergy_endpoint_reset()

UINT ux_hcd_synergy_endpoint_reset (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will reset an endpoint.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	Endpoint reset successfully.

◆ ux_hcd_synergy_entry()

UINT ux_hcd_synergy_entry (UX_HCD * *hcd*, UINT *function*, VOID * *parameter*)

This function is the entry function to the USB driver from the USB stack.

Parameters

[in]	hcd	: Pointer to USBX Host Controller structure.
[in]	function	: Function for driver to perform
[in]	parameter	: Pointer to function parameter(s)

Return values

UX_SUCCESS	HCD function is dispatched successfully.
UX_CONTROLLER_UNKNOWN	Synergy controller is not known.
UX_FUNCTION_NOT_SUPPORTED	Function not supported.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_controller_disable()
- ux_hcd_synergy_port_status_get()
- ux_hcd_synergy_port_enable()
- ux_hcd_synergy_port_disable()
- ux_hcd_synergy_power_on_port()
- ux_hcd_synergy_power_down_port()
- ux_hcd_synergy_port_suspend()
- ux_hcd_synergy_port_resume()
- ux_hcd_synergy_port_reset()
- ux_hcd_synergy_frame_number_get()
- ux_hcd_synergy_request_transfer()
- ux_hcd_synergy_transfer_abort()
- ux_hcd_synergy_control_endpoint_create()
- ux_hcd_synergy_bulk_endpoint_create()
- ux_hcd_synergy_interrupt_endpoint_create()
- ux_hcd_synergy_isochronous_endpoint_create()
- ux_hcd_synergy_asynchronous_endpoint_destroy()
- ux_hcd_synergy_periodic_endpoint_destroy()
- ux_hcd_synergy_endpoint_reset()

◆ ux_hcd_synergy_fifo_port_change()

ULONG ux_hcd_synergy_fifo_port_change (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*, ULONG *direction*)

This function change the port of the FIFO.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure
[in]	direction	: Direction to transfer

Return values

synergy_register	Content of FIFO control register.
UX_ERROR	Port not changed successfully or unable to access FIFO.

◆ ux_hcd_synergy_fifo_read()

UINT ux_hcd_synergy_fifo_read (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function read data from the FIFO configured for the PIPE(FIFO C, D0 or D1).

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure

Return values

UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_READ_OVER	Status set to read overflow.
UX_SYNERGY_HC_FIFO_READ_SHORT	Short packet to read.
UX_SYNERGY_HC_FIFO_READING	Continue reading buffer.

◆ ux_hcd_synergy_fifo_write_software_copy()

VOID ux_hcd_synergy_fifo_write_software_copy (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *payload_length*, UCHAR * *payload_buffer*, VOID * *fifo_addr*, ULONG *fifo_sel*)

USBX HCD CPU FIFO write by software copy. Call a suitable subroutine for selected USB controller hardware.

Parameters

[in]	hcd_synergy	Pointer to the HCD control block
[in]	payload_length	Payload length
[in]	payload_buffer	Payload buffer address
[in]	fifo_addr	FIFO register address
[in]	fifo_sel	FIFO select register

◆ ux_hcd_synergy_fifo_write_software_copy_remaining_bytes()

VOID ux_hcd_synergy_fifo_write_software_copy_remaining_bytes (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *payload_length*, UCHAR * *payload_buffer*, VOID * *fifo_addr*)

USBX HCD CPU FIFO write - Copy remaining bytes to FIFO by software if the rest bytes are less than FIFO access width.

Parameters

[in]	hcd_synergy	Pointer to the HCD control block
[in,out]	payload_length	Payload length
[in,out]	payload_buffer	Payload buffer address
[in,out]	fifo_addr	FIFO register address

◆ ux_hcd_synergy_fifoc_write()

UINT ux_hcd_synergy_fifoc_write (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function writes a buffer to FIFOC.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to Synergy ED structure

Return values

UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_WRITE_END	Writing at ends.
UX_SYNERGY_HC_FIFO_WRITE_SHORT	Writing short data.
UX_SYNERGY_HC_FIFO_WRITING	Doing multiple writes.

◆ ux_hcd_synergy_fifod_write()

UINT ux_hcd_synergy_fifod_write (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function writes a buffer data to FIFOD0 or FIFOD1.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to Synergy ED structure

Return values

UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_WRITE_END	Writing at ends.
UX_SYNERGY_HC_FIFO_WRITE_SHORT	Writing short data.
UX_SYNERGY_HC_FIFO_WRITING	Doing multiple writes.

◆ ux_hcd_synergy_frame_number_get()

UINT ux_hcd_synergy_frame_number_get (UX_HCD_SYNERGY * *hcd_synergy*, ULONG * *frame_number*)

This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes and for timing.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	frame_number	: Frame number to set

Return values

UX_SUCCESS	Frame number returned successfully.

◆ ux_hcd_synergy_frame_number_set()

VOID ux_hcd_synergy_frame_number_set (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *frame_number*)

This function will set the current frame number to the one specified. This function is mostly used for isochronous purpos.es.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	frame_number	: Frame number to set

◆ ux_hcd_synergy_initialize()

UINT ux_hcd_synergy_initialize (UX_HCD * *hcd*)

This function initializes the Synergy controller.

Parameters

[in,out]	hcd	: Pointer to USBX host controller structure.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_initialize_common()

◆ ux_hcd_synergy_initialize_transfer_support()

UINT ux_hcd_synergy_initialize_transfer_support (UX_HCD * *hcd*, const UX_HCD_SYNERGY_TRANSFER * *p_transfer_instance*)

USBX HCD Transfer Support with DMA support.

Parameters

[in,out]	hcd	Pointer to the USBX HCD control block.
[in]	p_transfer_instance	Pointer to Transfer module instances.

Return values

UX_SUCCESS	Initialize hcd transfer support successfully.
UX_CONTROLLER_INIT_FAILED	Failed in Transfer module setup, or Unsupported USB controller was specified.
UX_SEMAPHORE_ERROR	Failed in creating a semaphore used for DMA transfer.
UX_MEMORY_INSUFFICIENT	Failed in allocation memory.

◆ ux_hcd_synergy_interrupt_endpoint_create()

UINT ux_hcd_synergy_interrupt_endpoint_create (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will create an interrupt endpoint. The interrupt endpoint has an interval of operation from 1 to 255. The Synergy has no hardware scheduler but we still build an interrupt tree similar to the OHCI controller.

This routine will match the best interval for the Synergy hardware. It will also determine the best node to hook the endpoint based on the load that already exists on the horizontal ED chain.

The tricky part is to understand how the interrupt matrix is constructed. We have used eds with the skip bit on to build a frame of anchor eds. Each ED creates a node for an appropriate combination of interval frequency in the list.

After obtaining a pointer to the list with the lowest traffic, we traverse the list from the highest interval until we reach the interval required. At that node, we anchor our real ED to the node and link the ED that was attached to the node to our ED.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	Interrupt endpoint created successfully.
UX_ERROR	Available pipe is not found for interrupt endpoint.
UX_NO_ED_AVAILABLE	Failed to obtain an ED for new endpoint.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for terminating the ED transfer chain.

◆ ux_hcd_synergy_interrupt_handler()

VOID ux_hcd_synergy_interrupt_handler (UINT *hcd_index*)

This function is the interrupt handler for the Synergy USB HS controller. Normally an interrupt occurs from the controller when there is either a EOF signal and there has been transfers within the frame or when there is a change on one of the downstream ports.

All we need to do in the ISR is scan the controllers to find out which one has issued a IRQ. If there is work to do for this controller we need to wake up the corresponding thread to take care of the job.

Parameters

[in]	hcd_index	: HCD number

Check if the controller is operational, if not, skip it.

Examine the source of interrupts. Check for SOF signal.

Check for Over Current condition.

Check if we have a BEMP interrupt.

Do we have a BRDY interrupt ?

Do we have a NRDY interrupt ?

Check for attach signal.

Is it a detach signal ?

Check for BCHG signal.

Is it a EOFERR signal.

Did we get a SACK interrupt ?

Did we get a SIGN interrupt ?

◆ ux_hcd_synergy_iso_queue_process()

VOID ux_hcd_synergy_iso_queue_process (UX_HCD_SYNERGY * *hcd_synergy*)

This function process the isochronous transactions that happened in the last frame.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_iso_queue_process_bemp()

VOID ux_hcd_synergy_iso_queue_process_bemp (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the BEMP(Buffer Empty) Interrupt that occurred on a specific ED used for Isochronous transfer.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_iso_queue_process_brdy()

VOID ux_hcd_synergy_iso_queue_process_brdy (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED used for isochronous transfer.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_iso_queue_process_nrdy()

VOID ux_hcd_synergy_iso_queue_process_nrdy (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED used for Isochronous transfer.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

◆ ux_hcd_synergy_iso_schedule()

VOID ux_hcd_synergy_iso_schedule (UX_HCD_SYNERGY * *hcd_synergy*)

This function schedules new transfers from isochronous list.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_iso_td_add()

UINT ux_hcd_synergy_iso_td_add (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function adds a transfer descriptor to an Isochronous Endpoint Descriptor.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	ed	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	A transfer descriptor was added to an Isochronous Endpoint Descriptor successfully.
Others	See Common Error Codes for other possible return codes or causes. This function calls : ux_hcd_synergy_buffer_write()

◆ ux_hcd_synergy_isochronous_endpoint_create()

UINT ux_hcd_synergy_isochronous_endpoint_create (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function creates an isochronous endpoint.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	Isochronous endpoint is created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED terminating the ED transfer chain.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for new endpoint.

◆ ux_hcd_synergy_isochronous_td_obtain()

UX_SYNERGY_ISO_TD* ux_hcd_synergy_isochronous_td_obtain (UX_HCD_SYNERGY * *hcd_synergy*)

This function obtains a free TD from the isochronous TD list.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

td	Pointer to Synergy ISO Trasfer Descriptor.
UX_NULL	TD not available in the TD list.

◆ ux_hcd_synergy_least_traffic_list_get()

UX_SYNERGY_ED* ux_hcd_synergy_least_traffic_list_get (UX_HCD_SYNERGY * *hcd_synergy*)

This function return a pointer to the first ED in the periodic tree that has the least traffic registered.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

min_bandwidth_ed	End descriptor interrupt Number(ed)

◆ ux_hcd_synergy_periodic_endpoint_destroy()

UINT ux_hcd_synergy_periodic_endpoint_destroy (UX_HCD_SYNERGY * *hcd_synergy*, UX_ENDPOINT * *endpoint*)

This function will destroy an isochronous endpoint.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	endpoint	: Pointer to an Endpoint control block

Return values

UX_SUCCESS	Isochronous endpoint is destroyed successfully.
UX_ENDPOINT_HANDLE_UNKNOWN	Physical endpoint has not been initialized properly.

◆ ux_hcd_synergy_periodic_schedule()

VOID ux_hcd_synergy_periodic_schedule (UX_HCD_SYNERGY * *hcd_synergy*)

This function schedules new transfers from the periodic interrupt list.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_periodic_tree_create()

UINT ux_hcd_synergy_periodic_tree_create (UX_HCD_SYNERGY * *hcd_synergy*)

This function creates the periodic static tree for the interrupt and isochronous eds.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

UX_SUCCESS	Periodic tree created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED.

◆ ux_hcd_synergy_port_disable()

UINT ux_hcd_synergy_port_disable (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will disable a specific port attached to the root HUB.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_SUCCESS	Port disabled successfully.
UX_PORT_INDEX_UNKNOWN	Invalid port .

◆ ux_hcd_synergy_port_enable()

UINT ux_hcd_synergy_port_enable (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will enable a specific port attached to the root HUB.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_SUCCESS	Port enabled successfully.
UX_PORT_INDEX_UNKNOWN	Invalid port.
UX_NO_DEVICE_CONNECTED	Device not connected properly.

◆ ux_hcd_synergy_port_reset()

UINT ux_hcd_synergy_port_reset (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will reset a specific port attached to the root HUB.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_SUCCESS	Port reset successfully.
UX_PORT_INDEX_UNKNOWN	Invalid port.
UX_NO_DEVICE_CONNECTED	Device not connected properly.

◆ ux_hcd_synergy_port_resume()

UINT ux_hcd_synergy_port_resume (UX_HCD_SYNERGY * *hcd_synergy*, UINT *port_index*)

This function will resume a specific port attached to the root HUB. Present, this function is not supported for resume port.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

◆ ux_hcd_synergy_port_status_get()

ULONG ux_hcd_synergy_port_status_get (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will return the status for each port attached to the root HUB.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_PORT_INDEX_UNKNOWN	Invalid port.
port_status	Synergy Port status

Check to see if this port is valid on this controller.

The port is valid, build the status mask for this port. This function returns a controller agnostic bit field.

Provides a delay of 100 ms to stabilize while initial power up

Find the number of enumeration events occurred on another HOST.

Wait for the semaphore to be put by the root hub or a regular hub.

◆ ux_hcd_synergy_port_suspend()

UINT ux_hcd_synergy_port_suspend (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will suspend a specific port attached to the root HUB. Present, this function is does not supported.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

◆ ux_hcd_synergy_power_down_port()

UINT ux_hcd_synergy_power_down_port (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will power down a specific port attached to the root HUB. Present, this function is does not supported.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

◆ ux_hcd_synergy_power_on_port()

UINT ux_hcd_synergy_power_on_port (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *port_index*)

This function will power a specific port attached to the root HUB. Present, this function is does not supported.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	port_index	: Port Index

Return values

UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

◆ ux_hcd_synergy_power_root_hubs()

VOID ux_hcd_synergy_power_root_hubs (UX_HCD_SYNERGY * *hcd_synergy*)

This function will power the root HUB. Present, this function is does not supported.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block

◆ ux_hcd_synergy_register_clear()

VOID ux_hcd_synergy_register_clear (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function clears flags in a synergy USB register.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	synergy_register	: Register to write
[in]	value	: Value to clear

◆ ux_hcd_synergy_register_read()

ULONG ux_hcd_synergy_register_read (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *synergy_register*)

This function reads a data from synergy USB register.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	synergy_register	: Register to read

Return values

hcd_reg	Value read from the specified register(ULONG value).

◆ ux_hcd_synergy_register_set()

VOID ux_hcd_synergy_register_set (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function sets flags in a synergy USB register.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	synergy_register	: Register to read
[in]	value	: Value to be set

◆ ux_hcd_synergy_register_status_clear()

VOID ux_hcd_synergy_register_status_clear (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function clears a bit in a status register of the synergy controller.To clear the status bits, need to write 0 only to the bits to be cleared. Write 1 to the other bits.

Parameters

[in,out]	hcd_synergy	: Pointer to a DCD control block
[in,out]	synergy_register	: Register to clear
[in,out]	value	: Value to clear

◆ ux_hcd_synergy_register_write()

VOID ux_hcd_synergy_register_write (UX_HCD_SYNERGY * *hcd_synergy*, ULONG *synergy_register*, USHORT *value*)

This function writes a data to a Synergy USB register.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in]	synergy_register	: Register to write
[in]	value	: Value to write

◆ ux_hcd_synergy_regular_td_obtain()

UX_SYNERGY_TD* ux_hcd_synergy_regular_td_obtain (UX_HCD_SYNERGY * *hcd_synergy*)

This function obtains a free TD from the regular TD list.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block

Return values

td	A pointer to Synergy TD.
UX_NULL	Null pointer.

◆ ux_hcd_synergy_request_bulk_transfer()

UINT ux_hcd_synergy_request_bulk_transfer (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function performs a bulk transfer request. A bulk transfer can be larger than the size of the Synergy buffer so it may be required to chain multiple tds to accommodate this transfer request. A bulk transfer is non blocking, so we return before the transfer request is completed.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Bulk transfer happened successfully.
UX_NO_TD_AVAILABLE	Unavailable New TD.

Perform bulk transfer. If transfer request payload length is zero, transfer it once.

◆ ux_hcd_synergy_request_control_transfer()

UINT ux_hcd_synergy_request_control_transfer (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function performs a control transfer from a transfer request. The USB control transfer is in 3 phases (setup, data, status). This function will chain all phases of the control sequence before setting the Synergy endpoint as a candidate for transfer.

Parameters

[in,out]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_MEMORY_INSUFFICIENT	Insufficient memory to build the SETUP request.
UX_NO_TD_AVAILABLE	Unavailable New TD.
ux_transfer_request_completion_code	Pointer to USBX transfer request structure(request completion code).

◆ ux_hcd_synergy_request_interrupt_transfer()

UINT ux_hcd_synergy_request_interrupt_transfer (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function performs an interrupt transfer request. An interrupt transfer can only be as large as the MaxpacketField in the endpoint descriptor. This was verified at the USB layer and does not need to be reverified here.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Interrupt transfer request processed successfully.
UX_NO_TD_AVAILABLE	Unavailable new TD.

◆ ux_hcd_synergy_request_isochronous_transfer()

UINT ux_hcd_synergy_request_isochronous_transfer (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function performs an isochronous transfer request.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Isochronous transfer request processed successfully.
UX_NO_TD_AVAILABLE	Unavailable new TD.

◆ ux_hcd_synergy_request_transfer()

UINT ux_hcd_synergy_request_transfer (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function is the handler for all the transactions on the USB. The transfer request passed as parameter contains the endpoint and the device descriptors in addition to the type of transaction de be executed. This function routes the transfer request to according to the type of transfer to be executed.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_ERROR	Error while Isolating the endpoint type and routing the transfer request.
UX_NO_DEVICE_CONNECTED	No device attached.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_request_control_transfer()
- ux_hcd_synergy_request_bulk_transfer()
- ux_hcd_synergy_request_interrupt_transfer()
- ux_hcd_synergy_request_isochronous_transfer()

◆ ux_hcd_synergy_td_add()

UINT ux_hcd_synergy_td_add (UX_HCD_SYNERGY * *hcd_synergy*, UX_SYNERGY_ED * *ed*)

This function add new TD for control, Bulk or Interrupt endpoint.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in]	ed	: Pointer to Synergy ED structure

Return values

UX_SUCCESS	Transfer done successfully.

◆ ux_hcd_synergy_transfer_abort()

UINT ux_hcd_synergy_transfer_abort (UX_HCD_SYNERGY * *hcd_synergy*, UX_TRANSFER * *transfer_request*)

This function will abort transactions attached to a transfer request.

Parameters

[in]	hcd_synergy	: Pointer to a HCD control block
[in,out]	transfer_request	: Pointer to transfer request

Return values

UX_SUCCESS	Transactions attached to transfer request are aborted successfully.
UX_ENDPOINT_HANDLE_UNKNOWN	Endpoint is not initialized properly.

◆ ux_hcd_synergy_uninitialize()

UINT ux_hcd_synergy_uninitialize (ULONG *ux_hcd_io*)

This function un-initializes the Synergy HOST controller.

Parameters

[in]	ux_hcd_io	: HCD controller base address

Return values

UX_SUCCESS	Completed the USB controller Un-initialization successfully.
UX_SYNERGY_UNINIT_FAILED	HCD controller un-initialization failed.

Returns

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_uninitialize_transfer_support()

Reset the BEMPE, NRDYE, BRDYE, SOFE bits.

Clear the INTENB1 bits.

Reset the BRDY, NRDY, BEMPE for all pipes.

Clear USB interrupt status0 register.

Clear USB interrupt status1 register.

Clear all the physical endpoint.

Disable pull-up/pull-down of the D+/D- line.

Disable USB clock operation.

uninitialize and disable DMA support

Stop the module usage

◆ ux_hcd_synergy_uninitialize_transfer_support()

UINT ux_hcd_synergy_uninitialize_transfer_support (UX_HCD_SYNERGY * *hcd_synergy*)
This function un-initializes the transfer module associated with the USBX HOST controller.

Parameters

[in]	hcd_synergy	: HCD synergy controller instance.

Return values

UX_SUCCESS	Completed the transfer Un-initialization successfully.
UX_SYNERGY_UNINIT_FAILED	Failed to Un-initialize the USB controller.

UX_DCD_SYNERGY_ED Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_dcd_synergy.h>

Data Fields

TX_SEMAPHORE	ux_dcd_synergy_ep_slave_transfer_request_semaphore

Detailed Description

Define USB SYNERGY physical endpoint structure.

Field Documentation

◆ ux_dcd_synergy_ep_slave_transfer_request_semaphore

TX_SEMAPHORE UX_DCD_SYNERGY_ED::ux_dcd_synergy_ep_slave_transfer_request_semaphore
Each pipe to have its own transfer request semaphore

The documentation for this struct was generated from the following file:

- ux_dcd_synergy.h

UX_DCD_SYNERGY_TRANSFER Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_dcd_synergy.h>

Detailed Description

Define SYNERGY Transfer structure

The documentation for this struct was generated from the following file:

- ux_dcd_synergy.h

UX_DCD_SYNERGY_PAYLOAD_TRANSFER Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_dcd_synergy.h>

Data Fields

UCHAR *	payload_buffer Destination buffer address.
ULONG	payload_length Payload length to transmit.
UINT	transfer_times Number of transfer.
UINT	transfer_width Bytes per transfer.
UINT	transfer_blocks Number of blocks of above width * times.

Detailed Description

Define SYNERGY Payload Transfer structure

The documentation for this struct was generated from the following file:

- ux_dcd_synergy.h

UX_DCD_SYNERGY Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_dcd_synergy.h>

Data Fields

ULONG	ux_dcd_synergy_pipe [UX_DCD_SYNERGY_MAX_PIPE+1] "+1" is added to support total number of pipes e.g.S1 series supports 0-7(total 8) pipes, as "UX_DCD_SYNERGY_MAX_PIPE" is defined as 7
UINT	ux_dcd_synergy_D0_fifo_state
UINT	ux_dcd_synergy_D1_fifo_state

Detailed Description

Define USB SYNERGY DCD structure definition.

Field Documentation

◆ ux_dcd_synergy_D0_fifo_state

UINT UX_DCD_SYNERGY::ux_dcd_synergy_D0_fifo_state
This variable determines the D0 FIFO is used for DMA or not

◆ ux_dcd_synergy_D1_fifo_state

UINT UX_DCD_SYNERGY::ux_dcd_synergy_D1_fifo_state
This variable determines the D1 FIFO is used for DMA or not

The documentation for this struct was generated from the following file:

- ux_dcd_synergy.h

UX_HCD_SYNERGY_TRANSFER Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Detailed Description

Define synergy transfer structure

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_HCD_SYNERGY Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Data Fields

ULONG	ux_synergy_next_available_bufnum will need to implement some type of dynamic buffer management, for now just carve off */

Detailed Description

Define Synergy structure.

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_HCD_SYNERGY_PAYLOAD_TRANSFER Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Data Fields

UCHAR *	payload_buffer Destination buffer address.
ULONG	payload_length Payload length to transmit.
UCHAR	transfer_width Bytes per transfer.

Detailed Description

Define SYNERGY Payload Transfer structure

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_HCD_SYNERGY_FIFO Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Data Fields

ULONG	fifo_sel FIFO type.
VOID *	fifo_addr

	Selected FIFO address.
ULONG	fifo_ctrl Selected FIFO control.

Detailed Description

Define SYNERGY fifo structure

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_SYNERGY_ED Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Detailed Description

Define Synergy ED structure.

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_SYNERGY_TD Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Detailed Description

Define Synergy TD structure.

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

UX_SYNERGY_ISO_TD Struct Reference

Renesas Synergy Software Package Reference » Framework Layer » USBX Framework

#include <ux_hcd_synergy.h>

Detailed Description

Define Synergy ISOCHRONOUS TD structure.

The documentation for this struct was generated from the following file:

- ux_hcd_synergy.h

5.1.3.53 2D Drawing Engine Support Framework

Renesas Synergy Software Package Reference » Framework Layer

Functions

d1_device *	d1_opendevice (d1_long_t flags) Creates a device handle to access hardware. This function initializes the D1 device handle, supply module clock to D/AVE 2D hardware and enables the D/AVE 2D interrupt. It is called by the D/AVE 2D driver function d2_inithw() to initialize the D/AVE 2D hardware. More...
d1_int_t	d1_closedevice (d1_device *handle) Close a device handle. It is called by the D/AVE 2D driver function d2_deinithw to de-initialize the D/AVE 2D hardware. Disables the D/AVE 2D interrupt and stop the module clock supply. More...
void	d1_setregister (d1_device *handle, d1_int_t deviceid, d1_int_t index, d1_long_t value) Write data to the D/AVE 2D hardware register. More...
d1_long_t	d1_getregister (d1_device *handle, d1_int_t deviceid, d1_int_t index) Read data from hardware register. Reading a register from an invalid or unsupported device ID will always return 0. More...

d1_int_t	d1_devicesupported (d1_device *handle, d1_int_t deviceid) Check if the specified device ID is valid for the D/AVE 2D implemented for Synergy. Use this function to verify that a specific hardware interface is available on the current host system. More...
void	drw_int_isr (void) DRW ISR. More...
d1_int_t	d1_initirq_intern (d1_device_synergy *handle) Initialize IRQ for D/AVE 2D hardware. More...
d1_int_t	d1_shutdownirq_intern (d1_device_synergy *handle) De-initialize IRQ for D/AVE 2D hardware. More...
d1_int_t	d1_queryirq (d1_device *handle, d1_int_t irqmask, d1_int_t timeout) Wait for next IRQ being happened. More...
void *	d1_allocmem (d1_uint_t size) Allocates memory in the driver heap. More...
void	d1_freemem (void *ptr) Free the specified memory area in the driver heap. More...
d1_uint_t	d1_memsize (void *ptr) This function intends to return the size of the given memory block but we don't return valid value. This function returns always 1. More...
void *	d1_allocvidmem (d1_device *handle, d1_int_t memtype, d1_uint_t size) Allocate video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_allocmem. More...
void	d1_freevidmem (d1_device *handle, d1_int_t memtype, void *ptr) Free video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_freemem. More...

d1_int_t	d1_queryvidmem (d1_device *handle, d1_int_t memtype, d1_int_t query) Get current memory status. This function does not do anything special. More...
d1_int_t	d1_queryarchitecture (d1_device *handle) Return hints about systems memory architecture. More...
void *	d1_mapvidmem (d1_device *handle, void *ptr, d1_int_t flags) Map video memory for direct CPU access. Synergy does not use the virtual memory space so this function does not do anything special. More...
d1_int_t	d1_unmapvidmem (d1_device *handle, void *ptr) Release memory mapping. Synergy does not use the virtual memory space so this function does not do anything special. More...
void *	d1_maptovidmem (d1_device *handle, void *ptr) Map CPU accessible address of a video memory block back to video memory address. Synergy does not use the virtual memory space so this function does not do anything special. More...
void *	d1_mapfromvidmem (d1_device *handle, void *ptr) Map already allocated video memory address to an address for direct CPU access. More...
d1_int_t	d1_copytovidmem (d1_device *handle, void *dst, const void *src, d1_uint_t size, d1_int_t flags) Copy data to video memory. Destination (video) memory area has to be allocated by d1_allocvidmem. More...
d1_int_t	d1_copyfromvidmem (d1_device *handle, void *dst, const void *src, d1_uint_t size, d1_int_t flags) Copy data from video memory. Source (video) memory area has to be allocated by d1_allocvidmem. More...
d1_int_t	d1_cacheflush (d1_device *handle, d1_int_t memtype) Flush CPU data caches. Synergy does not have a cache memory so

	does not do anything special. More...
d1_int_t	d1_cacheblockflush (d1_device *handle, d1_int_t memtype, const void *ptr, d1_uint_t size) Flush part of CPU data caches. Synergy does not have a cache memory so does not do anything special. More...

Detailed Description

Function Documentation

◆ d1_allocmem()

void* d1_allocmem (d1_uint_t *size*)

Allocates memory in the driver heap.

Parameters

[in]	size	Size of the memory to be allocated.

Return values

Non-NULL	The function returns the pointer to memory chunk if memory allocation was successful.
NULL	The function returns NULL if memory allocation was failed.

Create a byte memory pool in the driver heap if this function call is the first time.

Allocate memory from a byte memory pool.

◆ d1_allocvidmem()

void* d1_allocvidmem (d1_device * *handle*, d1_int_t *memtype*, d1_uint_t *size*)

Allocate video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_allocmem.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	memtype	Type of memory.
[in]	size	Number of bytes.

Return values

Non-NULL	The function returns the pointer to memory chunk if memory allocation was successful.
NULL	The function returns Null if memory allocation was failed.

◆ d1_cacheblockflush()

d1_int_t d1_cacheblockflush (d1_device * *handle*, d1_int_t *memtype*, const void * *ptr*, d1_uint_t *size*)

Flush part of CPU data caches. Synergy does not have a cache memory so does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	memtype	Memory pools to flush (can be ored together).
[in]	ptr	Start address of memory to be flushed.
[in]	size	Size of memory to be flushed.

Return values

1	The function always return 1.

◆ d1_cacheflush()

d1_int_t d1_cacheflush (d1_device * *handle*, d1_int_t *memtype*)

Flush CPU data caches. Synergy does not have a cache memory so does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	memtype	Memory pools to flush (can be ored together).

Return values

1	The function always return 1.

◆ d1_closedevice()

d1_int_t d1_closedevice (d1_device * *handle*)

Close a device handle. It is called by the D/AVE 2D driver function d2_deinithw to de-initialize the D/AVE 2D hardware. Disables the D/AVE 2D interrupt and stop the module clock supply.

Parameters

[in]	handle	Pointer to the d1_device object.

Return values

0	The function returns 0 if error occurred, NULL is passed to the argument handle.
1	The function returns 1 if successfully device handle was closed.

Disable the D/AVE 2D interrupt.

Stop clock supply to the D/AVE 2D hardware.

Delete used semaphore

◆ d1_copyfromvidmem()

d1_int_t d1_copyfromvidmem (d1_device * *handle*, void * *dst*, const void * *src*, d1_uint_t *size*, d1_int_t *flags*)

Copy data from video memory. Source (video) memory area has to be allocated by d1_allocvidmem.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	dst	pointer into system memory (destination).
[in]	src	Pointer into video memory (source).
[in]	size	Number of bytes to copy.
[in]	flags	Reserved for future use.

Return values

1	The function always return 1.

Simply use C standard memcpy.

◆ d1_copytovidmem()

d1_int_t d1_copytovidmem (d1_device * *handle*, void * *dst*, const void * *src*, d1_uint_t *size*, d1_int_t *flags*)

Copy data to video memory. Destination (video) memory area has to be allocated by d1_allocvidmem.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	dst	pointer into video memory (destination).
[in]	src	Pointer into system memory (source).
[in]	size	Number of bytes to copy.
[in]	flags	Bitfield containing additional information on data to be copied.

Return values

1	The function always return 1.

Simply use C standard memcpy.

◆ d1_devicesupported()

d1_int_t d1_devicesupported (d1_device * *handle*, d1_int_t *deviceid*)

Check if the specified device ID is valid for the D/AVE 2D implemented for Synergy. Use this function to verify that a specific hardware interface is available on the current host system.

Parameters

[in]	handle	Pointer to a device handle.
[in]	deviceid	D1_DAVE2D(Rendering core). The others are ignored.

Return values

0	The function returns 0 if specified device ID not supported.
1	The function returns 1 if specified device ID supported.

Check the deviceid.

Return 1 in case of valid deviceid.

Return 0 in case of Unknown device ID.

◆ d1_freemem()

void d1_freemem (void * *ptr*)

Free the specified memory area in the driver heap.

Parameters

[in]	ptr	Pointer to the memory area to be freed.

Free specified memory allocated by d1_allocmem.

◆ d1_freevidmem()

void d1_freevidmem (d1_device * *handle*, d1_int_t *memtype*, void * *ptr*)

Free video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_freemem.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	memtype	Type of memory.
[in]	ptr	Address returned by d1_allocvidmem.

◆ d1_getregister()

d1_long_t d1_getregister (d1_device * *handle*, d1_int_t *deviceid*, d1_int_t *index*)

Read data from hardware register. Reading a register from an invalid or unsupported device ID will always return 0.

Parameters

[in]	handle	Pointer to a device handle.
[in]	deviceid	D1_DAVE2D(Rendering core) or D1_DLISTINDIRECT(Lists of dlist support). The others are ignored.
[in]	index	Register index (starts with 0).

Return values

Value	The function returns 32-bit value of the register.

Check the deviceid to see whether Register addressing or dlist mode is being used.

If Register addressing is used return the value stored in specified D/AVE 2D register.

If dlist mode is used return the dlist address.

If neither Register addressing nor dlist mode is used return 0.

◆ d1_initirq_intern()

d1_int_t d1_initirq_intern (d1_device_synergy * *handle*)

Initialize IRQ for D/AVE 2D hardware.

Parameters

[in]	handle	Pointer to the d1_device object.

Return values

0	The function returns 0 if failed in the IRQ initialization.
1	The function returns 1 if successfully IRQ is initialized.

Clear all the D/AVE 2D IRQs and enable Display list IRQ.

Access to FMI through the eventInfoGet interface and get the D/AVE 2D IRQ information.

Check return value of eventInfoGet API, if SUCCESS Enable the D/AVE 2D IRQ Vector

Set D/AVE 2D interrupt priority in NVIC if the IRQ vector is valid.

Enable D/AVE 2D interrupt in NVIC if the IRQ vector is valid.

◆ d1_mapfromvidmem()

void* d1_mapfromvidmem (d1_device * *handle*, void * *ptr*)

Map already allocated video memory address to an address for direct CPU access.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	ptr	Video memory address returned by d1_allocvidmem.

Return values

ptr	The function just returns ptr back since no mapping required for Synergy.

◆ d1_maptovidmem()

void* d1_maptovidmem (d1_device * *handle*, void * *ptr*)

Map CPU accessible address of a video memory block back to video memory address. Synergy does not use the virtual memory space so this function does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	ptr	CPU accessible address pointing to a video memory block originally allocated using d1_allocvidmem.

Return values

ptr	The function just returns ptr back since no mapping required for Synergy.

◆ d1_mapvidmem()

void* d1_mapvidmem (d1_device * *handle*, void * *ptr*, d1_int_t *flags*)

Map video memory for direct CPU access. Synergy does not use the virtual memory space so this function does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	ptr	Video memory address returned by d1_allocvidmem.
[in]	flags	Memory mapping flags.

Return values

ptr	The function just returns ptr back since no mapping required for Synergy.

◆ d1_memsize()

d1_uint_t d1_memsize (void * *ptr*)

This function intends to return the size of the given memory block but we don't return valid value. This function returns always 1.

Parameters

[in]	ptr	Pointer to a memory block in Heap.

Return values

1	The function always return 1.

Always return 1.

◆ d1_opendevice()

d1_device* d1_opendevice (d1_long_t *flags*)

Creates a device handle to access hardware. This function initializes the D1 device handle, supply module clock to D/AVE 2D hardware and enables the D/AVE 2D interrupt. It is called by the D/AVE 2D driver function d2_inithw() to initialize the D/AVE 2D hardware.

Parameters

[in]	flags	Reserved. Not used in this function.

Return values

Non-NULL	The function returns the pointer to a d1_device object if the D1 device handle was successfully initialized.
NULL	The function returns NULL if failed to create the display list synchronization semaphore.

Get new device handle.

Initialize device data.

Access to FMI through the productFeatureGet interface and get the D/AVE 2D hardware information.

If failed to get the D/AVE 2D information from FMI, set NULL to handle and return.

Initialize the D/AVE 2D hardware bass address in the device context.

Supply clock to the D/AVE 2D hardware.

Create the semaphore to notify the completion of display list execution.

Enable the D/AVE 2D interrupt if semaphore creation was successful.

If failed to enable the D/AVE 2D interrupt, set NULL to handle.

If failed to create the semaphore, set NULL to handle.

Returns the pointer to the d1_device object.

◆ d1_queryarchitecture()

d1_int_t d1_queryarchitecture (d1_device * *handle*)

Return hints about systems memory architecture.

Parameters

[in]	handle	Pointer to the d1_device object.

Return values

d1_ma_unified	The function always return d1_ma_unified (Unified memory architecture).

Return d1_ma_unified as Memory architecture is such that CPU can directly access pointers returned by allocvidmem.

◆ d1_queryirq()

d1_int_t d1_queryirq (d1_device * *handle*, d1_int_t *irqmask*, d1_int_t *timeout*)

Wait for next IRQ being happened.

Parameters

[in]	handle	Pointer to the d1_device object (Not used).
[in]	irqmask	Interrupt ID (Not used. Synergy only uses Display list IRQ).
[in]	timeout	Timeout value.

Return values

0	The function returns 0 if wait through semaphore is not successful
1	The function returns 1 if wait through semaphore is successful.

Wait for dlist processing to complete.

◆ d1_queryvidmem()

d1_int_t d1_queryvidmem (d1_device * *handle*, d1_int_t *memtype*, d1_int_t *query*)

Get current memory status. This function does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	memtype	Type of memory.
[in]	query	Type of requested information.

Return values

0	The function always return 0.

◆ d1_setregister()

void d1_setregister (d1_device * *handle*, d1_int_t *deviceid*, d1_int_t *index*, d1_long_t *value*)

Write data to the D/AVE 2D hardware register.

Parameters

[in]	handle	Pointer to a device handle.
[in]	deviceid	D1_DAVE2D(Rendering core) or D1_DLISTINDIRECT(Lists of dlist support). The others are ignored.
[in]	index	Register index (word offset from the D/AVE 2D base address).
[in]	value	32-bit value to write.

Write data to specified D/AVE 2D register.

◆ d1_shutdownirq_intern()

d1_int_t d1_shutdownirq_intern (d1_device_synergy * *handle*)

De-initialize IRQ for D/AVE 2D hardware.

Parameters

[in]	handle	Pointer to the d1_device object.

Return values

0	The function returns 0 if failed in the IRQ de-initialization.
1	The function returns 1 if successfully IRQ is de-initialized.

Access to FMI through the eventInfoGet interface and get the D/AVE 2D IRQ information.

Check return value of eventInfoGet API, if SUCCESS disable the D/AVE 2D IRQ Vector

Disable D/AVE 2D interrupt in NVIC if the IRQ vector is valid.

Clear all the D/AVE 2D IRQs and disable Display list IRQ.

◆ d1_unmapvidmem()

d1_int_t d1_unmapvidmem (d1_device * *handle*, void * *ptr*)

Release memory mapping. Synergy does not use the virtual memory space so this function does not do anything special.

Parameters

[in]	handle	Pointer to the d1_device object.
[in]	ptr	Mapped video memory address returned by d1_mapvidmem.

Return values

1	The function always return 1.

◆ drw_int_isr()

void drw_int_isr (void)
DRW ISR. Just in case, check if the driver initialization done. Get D/AVE 2D interrupt status. Clear all the D/AVE 2D interrupts (keep the Display list interrupt enable). Display list interrupt? Signal semaphore for driver. Clear IRQ status.

5.1.4 HAL Interfaces

Renesas Synergy Software Package Reference

Modules

ADC Interface	Interface for A/D Converters.
Analog Connect Interface	Interface for analog connections.
CAC Interface	Interface for clock frequency accuracy measurements.
CAN Interface	Interface for CAN peripheral.
CGC Interface	Interface for clock generation.
COMPARATOR Interface	Interface for Comparators.

	CRC Interface Interface for cyclic redundancy checking.
	Crypto Interface Cryptographic algorithm APIs for encryption/decryption, signing/verification, and hashing.
	CTSU v2 Interface Interface for Capacitive Touch Controllers.
	DAC Interface Interface for D/A converters.
	Display Interface Interface for LCD panel displays.
	DOC Interface Interface for the Data Operation Circuit.
	events and peripheral definitions Interface for the Event Link Controller.
	External IRQ Interface Interface for detecting external interrupts.
	Flash Interface Interface for the flash controller.
	FMI Interface Interface for reading on-chip factory information.
	I2C Interface Interface for I2C communication.

	I2S Interface The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.
	Input Capture Interface Interface for sampling input signals for pulse width.
	I/O Port Interface Interface for accessing I/O ports and configuring I/O functionality.
	JPEG Decode Interface Interface for JPEG decode functions.
	JPEG Encode Interface Interface for JPEG encode functions.
	Key Matrix Interface Interface for key matrix functions.
	Low Power Modes V2 Interface Interface for accessing low power modes.
	Low Voltage Detection Interface This section defines the API for the LVD (Low Voltage Detection) Driver.
	OPAMP Interface Interface for Operational Amplifiers.
	PDC Interface Interface for PDC functions.
	PTP driver Interface Interface for PTP functions.

	PTPEDMAC driver Interface Interface for PTPEDMAC functions.
	Quad SPI Flash Interface Interface for accessing external SPI flash devices.
	RTC Interface Interface for accessing the Realtime Clock.
	SD/MMC Interface Interface for accessing SD, eMMC, and SDIO devices.
	SLCDC Interface Interface for Segment LCD controllers.
	SPI Interface Interface for SPI communications.
	Timer Interface Interface for timer functions.
	Transfer Interface Interface for data transfer functions.
	UART Interface Interface for UART communications.
	WDT Interface Interface for watch dog timer functions.

Detailed Description

The HAL Interfaces offer common APIs for functional use cases. They can be implemented by one or more HAL layer drivers. Framework Layer drivers connect to these HAL drivers through the Interface

Layer.

5.1.4.1 ADC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for A/D Converters. More...

Data Structures

struct	adc_sample_state_t
struct	adc_callback_args_t
struct	adc_info_t
struct	adc_channel_cfg_t
struct	adc_cfg_t
struct	adc_api_t
struct	adc_instance_t

Macros

#define	ADC_API_VERSION_MAJOR (2U)

Typedefs

typedef void	adc_ctrl_t

Enumerations

enum	adc_mode_t { ADC_MODE_SINGLE_SCAN = 0, ADC_MODE_GROUP_SCAN = 1, ADC_MODE_CONTINUOUS_SCAN = 2 }
enum	adc_resolution_t { ADC_RESOLUTION_12_BIT = 0, ADC_RESOLUTION_10_BIT = 1, ADC_RESOLUTION_8_BIT = 2, ADC_RESOLUTION_14_BIT = 3, ADC_RESOLUTION_16_BIT = 4, ADC_RESOLUTION_24_BIT = 5 }
enum	adc_pga_t { , SINGLE_INPUT_GAIN_1 = 0x0, SINGLE_INPUT_GAIN_2 = 0x1, SINGLE_INPUT_GAIN_3 = 0x2, SINGLE_INPUT_GAIN_4 = 0x3, SINGLE_INPUT_GAIN_5 = 0x4, SINGLE_INPUT_GAIN_6 = 0x5, SINGLE_INPUT_GAIN_7 = 0x6, SINGLE_INPUT_GAIN_8 = 0x7,

	SINGLE_INPUT_GAIN_9 = 0x8, SINGLE_INPUT_GAIN_10 = 0x9, SINGLE_INPUT_GAIN_11 = 0xA, SINGLE_INPUT_GAIN_12 = 0xB, SINGLE_INPUT_GAIN_13 = 0xC, SINGLE_INPUT_GAIN_14 = 0xD, SINGLE_INPUT_GAIN_15 = 0xE, DIFFERENTIAL_INPUT_GAIN_1 = 0x81, DIFFERENTIAL_INPUT_GAIN_2 = 0x95, DIFFERENTIAL_INPUT_GAIN_3 = 0xA9, DIFFERENTIAL_INPUT_GAIN_4 = 0xBB }
enum	adc_alignment_t { ADC_ALIGNMENT_RIGHT = 0x0000, ADC_ALIGNMENT_LEFT = 0x8000 }

enum	adc_add_t {
	ADC_ADD_OFF = 0, ADC_ADD_TWO = 1, ADC_ADD_THREE = 2, ADC_ADD_FOUR = 3,
	ADC_ADD_SIXTEEN = 0x05, ADC_ADD_AVERAGE_TWO = 0x81, ADC_ADD_AVERAGE_FOUR = 0x83, ADC_ADD_AVERAGE_EIGHT = 0X84,
	ADC_ADD_AVERAGE_SIXTEEN = 0x85
	}

enum	adc_clear_t { ADC_CLEAR_AFTER_READ_OFF = 0x0000, ADC_CLEAR_AFTER_READ_ON = 0x0020 }
enum	adc_trigger_t { ADC_TRIGGER_ASYNC_EXT_TRG0, ADC_TRIGGER_SYNC_ELC, ADC_TRIGGER_SOFTWARE }

enum	adc_sample_state_reg_t {
	ADC_SAMPLE_STATE_CHANNEL_0 = 0, ADC_SAMPLE_STATE_CHANNEL_1, ADC_SAMPLE_STATE_CHANNEL_2, ADC_SAMPLE_STATE_CHANNEL_3,
	ADC_SAMPLE_STATE_CHANNEL_4, ADC_SAMPLE_STATE_CHANNEL_5, ADC_SAMPLE_STATE_CHANNEL_6, ADC_SAMPLE_STATE_CHANNEL_7,
	ADC_SAMPLE_STATE_CHANNEL_8, ADC_SAMPLE_STATE_CHANNEL_9, ADC_SAMPLE_STATE_CHANNEL_10,
	ADC_SAMPLE_STATE_CHANNEL_11,
	ADC_SAMPLE_STATE_CHANNEL_12,
	ADC_SAMPLE_STATE_CHANNEL_13,
	ADC_SAMPLE_STATE_CHANNEL_14,
	ADC_SAMPLE_STATE_CHANNEL_15,
	ADC_SAMPLE_STATE_CHANNEL_16_TO_21 = -3,
	ADC_SAMPLE_STATE_CHANNEL_16_TO_20 = -3,
	ADC_SAMPLE_STATE_CHANNEL_16_TO_27 = -3,
	ADC_SAMPLE_STATE_TEMPERATURE = -2,
	ADC_SAMPLE_STATE_VOLTAGE = -1
	}

enum	adc_cb_event_t { ADC_EVENT_SCAN_COMPLETE, ADC_EVENT_SCAN_COMPLETE_GROUP_B, ADC_EVENT_CALIBRATION_COMPLETE, ADC_EVENT_CONVERSION_COMPLETE }
enum	adc_group_a_t { ADC_GROUP_A_PRIORITY_OFF = 0,

	ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER = 1, ADC_GROUP_A_GROUP_B_RESTART_SCAN = 3, ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN = 0x8001 }
enum	adc_voltage_reference_t { ADC_EXTERNAL_VOLTAGE = 0x00, ADC_INTERNAL_VREF_1_5V, ADC_INTERNAL_VREF_2_0V, ADC_INTERNAL_VREF_2_5V }
enum	adc_over_current_t { OVER_CURRENT_DETECTION_DISABLE = 0x00, OVER_CURRENT_DETECTION_ENABLE = 0x01 }
enum	adc_register_t { ADC_REG_CHANNEL_0 = 0, ADC_REG_CHANNEL_1 = 1, ADC_REG_CHANNEL_2 = 2, ADC_REG_CHANNEL_3 = 3, ADC_REG_CHANNEL_4 = 4, ADC_REG_CHANNEL_5 = 5, ADC_REG_CHANNEL_6 = 6, ADC_REG_CHANNEL_7 = 7, ADC_REG_CHANNEL_8 = 8, ADC_REG_CHANNEL_9 = 9, ADC_REG_CHANNEL_10 = 10, ADC_REG_CHANNEL_11 = 11, ADC_REG_CHANNEL_12 = 12, ADC_REG_CHANNEL_13 = 13, ADC_REG_CHANNEL_14 = 14, ADC_REG_CHANNEL_15 = 15, ADC_REG_CHANNEL_16 = 16, ADC_REG_CHANNEL_17 = 17, ADC_REG_CHANNEL_18 = 18, ADC_REG_CHANNEL_19 = 19, ADC_REG_CHANNEL_20 = 20, ADC_REG_CHANNEL_21 = 21, ADC_REG_CHANNEL_22 = 22, ADC_REG_CHANNEL_23 = 23, ADC_REG_CHANNEL_24 = 24, ADC_REG_CHANNEL_25 = 25, ADC_REG_CHANNEL_26 = 26, ADC_REG_CHANNEL_27 = 27, ADC_REG_TEMPERATURE = -3, ADC_REG_VOLT = -2 }

Detailed Description

Interface for A/D Converters.

Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

Implemented by: ADC Sigma Delta ADC (SDADC)

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

ADC Interface description: ADC Driver

Macro Definition Documentation

◆ ADC_API_VERSION_MAJOR

#define ADC_API_VERSION_MAJOR (2U)
Includes board and MCU related header files. Version Number of API.

Typedef Documentation

◆ adc_ctrl_t

typedef void adc_ctrl_t
ADC control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ adc_add_t

enum adc_add_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_ADD_OFF	Addition turned off for channels/sensors.
ADC_ADD_TWO	Add two samples.
ADC_ADD_THREE	Add three samples.
ADC_ADD_FOUR	Add four samples.
ADC_ADD_SIXTEEN	Add sixteen samples.
ADC_ADD_AVERAGE_TWO	Average two samples.
ADC_ADD_AVERAGE_FOUR	Average four samples.
ADC_ADD_AVERAGE_EIGHT	Average eight samples.
ADC_ADD_AVERAGE_SIXTEEN	Average sixteen samples.

◆ adc_alignment_t

enum adc_alignment_t	
ADC data alignment definitions	
Enumerator	
ADC_ALIGNMENT_RIGHT	Data alignment right.
ADC_ALIGNMENT_LEFT	Data alignment left.

◆ adc_cb_event_t

enum adc_cb_event_t	
ADC callback event definitions	
Enumerator	
ADC_EVENT_SCAN_COMPLETE	Normal/Group A scan complete.
ADC_EVENT_SCAN_COMPLETE_GROUP_B	Group B scan complete.
ADC_EVENT_CALIBRATION_COMPLETE	Calibration complete.
ADC_EVENT_CONVERSION_COMPLETE	Conversion complete.

◆ adc_clear_t

enum adc_clear_t	
ADC clear after read definitions	
Enumerator	
ADC_CLEAR_AFTER_READ_OFF	Clear after read off.
ADC_CLEAR_AFTER_READ_ON	Clear after read on.

◆ adc_group_a_t

enum adc_group_a_t	
ADC action for group A interrupts group B scan. This enumeration is used to specify the priority between Group A and B in group mode.	
Enumerator	
ADC_GROUP_A_PRIORITY_OFF	Group A ignored and does not interrupt ongoing group B scan.
ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER	Group A interrupts Group B(single scan) which restarts at next Group B trigger.
ADC_GROUP_A_GROUP_B_RESTART_SCAN	Group A interrupts Group B(single scan) which restarts immediately after Group A scan is complete.
ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN	Group A interrupts Group B(continuous scan) which continues scanning without a new Group B trigger.

◆ adc_mode_t

enum adc_mode_t	
ADC operation mode definitions	
Enumerator	
ADC_MODE_SINGLE_SCAN	Single scan - one or more channels.
ADC_MODE_GROUP_SCAN	Two trigger sources to trigger scan for two groups which contain one or more channels.
ADC_MODE_CONTINUOUS_SCAN	Continuous scan - one or more channels.

◆ adc_over_current_t

enum adc_over_current_t	
ADC reference voltage selection (Applicable for S1/S3 series MCU's only	
Enumerator	
OVER_CURRENT_DETECTION_DISABLE	ADC over current detection disable.
OVER_CURRENT_DETECTION_ENABLE	ADC over current detection enable.

◆ adc_pga_t

enum adc_pga_t	
ADC pga setting definitions	
Enumerator	
SINGLE_INPUT_GAIN_1	Single ended input Gain_1.
SINGLE_INPUT_GAIN_2	Single ended input Gain_2.
SINGLE_INPUT_GAIN_3	Single ended input Gain_3.
SINGLE_INPUT_GAIN_4	Single ended input Gain_4.
SINGLE_INPUT_GAIN_5	Single ended input Gain_5.
SINGLE_INPUT_GAIN_6	Single ended input Gain_6.
SINGLE_INPUT_GAIN_7	Single ended input Gain_7.
SINGLE_INPUT_GAIN_8	Single ended input Gain_8.
SINGLE_INPUT_GAIN_9	Single ended input Gain_9.
SINGLE_INPUT_GAIN_10	Single ended input Gain_10.
SINGLE_INPUT_GAIN_11	Single ended input Gain_11.
SINGLE_INPUT_GAIN_12	Single ended input Gain_12.
SINGLE_INPUT_GAIN_13	Single ended input Gain_13.
SINGLE_INPUT_GAIN_14	Single ended input Gain_14.
SINGLE_INPUT_GAIN_15	Single ended input Gain_15.
DIFFERENTIAL_INPUT_GAIN_1	Differential input Gain_1 ADPGADCR0 = 0x8, ADPGAGS0 = 0x1.
DIFFERENTIAL_INPUT_GAIN_2	Differential input Gain_2.
DIFFERENTIAL_INPUT_GAIN_3	Differential input Gain_3.
DIFFERENTIAL_INPUT_GAIN_4	Differential input Gain_4.

◆ adc_register_t

enum adc_register_t	
ADC registers used for the Read() argument	
Enumerator	
ADC_REG_CHANNEL_0	ADC channel 0.
ADC_REG_CHANNEL_1	ADC channel 1.
ADC_REG_CHANNEL_2	ADC channel 2.
ADC_REG_CHANNEL_3	ADC channel 3.
ADC_REG_CHANNEL_4	ADC channel 4.
ADC_REG_CHANNEL_5	ADC channel 5.
ADC_REG_CHANNEL_6	ADC channel 6.
ADC_REG_CHANNEL_7	ADC channel 7.
ADC_REG_CHANNEL_8	ADC channel 8.
ADC_REG_CHANNEL_9	ADC channel 9.
ADC_REG_CHANNEL_10	ADC channel 10.
ADC_REG_CHANNEL_11	ADC channel 11.
ADC_REG_CHANNEL_12	ADC channel 12.
ADC_REG_CHANNEL_13	ADC channel 13.
ADC_REG_CHANNEL_14	ADC channel 14.
ADC_REG_CHANNEL_15	ADC channel 15.
ADC_REG_CHANNEL_16	ADC channel 16.
ADC_REG_CHANNEL_17	ADC channel 17.
ADC_REG_CHANNEL_18	ADC channel 18.
ADC_REG_CHANNEL_19	ADC channel 19.
ADC_REG_CHANNEL_20	ADC channel 20.

ADC_REG_CHANNEL_21	ADC channel 21.
ADC_REG_CHANNEL_22	ADC channel 22.
ADC_REG_CHANNEL_23	ADC channel 23.
ADC_REG_CHANNEL_24	ADC channel 24.
ADC_REG_CHANNEL_25	ADC channel 25.
ADC_REG_CHANNEL_26	ADC channel 26.
ADC_REG_CHANNEL_27	ADC channel 27.
ADC_REG_TEMPERATURE	ADC channel temperature.
ADC_REG_VOLT	ADC channel volt.

◆ adc_resolution_t

enum adc_resolution_t	
ADC data resolution definitions	
Enumerator	
ADC_RESOLUTION_12_BIT	12 bit resolution
ADC_RESOLUTION_10_BIT	10 bit resolution
ADC_RESOLUTION_8_BIT	8 bit resolution
ADC_RESOLUTION_14_BIT	14 bit resolution
ADC_RESOLUTION_16_BIT	16 bit resolution
ADC_RESOLUTION_24_BIT	24 bit resolution

◆ adc_sample_state_reg_t

enum adc_sample_state_reg_t	
ADC sample state registers	
Enumerator	
ADC_SAMPLE_STATE_CHANNEL_0	Sample state register channel 0.

ADC_SAMPLE_STATE_CHANNEL_1	Sample state register channel 1.
ADC_SAMPLE_STATE_CHANNEL_2	Sample state register channel 2.
ADC_SAMPLE_STATE_CHANNEL_3	Sample state register channel 3.
ADC_SAMPLE_STATE_CHANNEL_4	Sample state register channel 4.
ADC_SAMPLE_STATE_CHANNEL_5	Sample state register channel 5.
ADC_SAMPLE_STATE_CHANNEL_6	Sample state register channel 6.
ADC_SAMPLE_STATE_CHANNEL_7	Sample state register channel 7.
ADC_SAMPLE_STATE_CHANNEL_8	Sample state register channel 8.
ADC_SAMPLE_STATE_CHANNEL_9	Sample state register channel 9.
ADC_SAMPLE_STATE_CHANNEL_10	Sample state register channel 10.
ADC_SAMPLE_STATE_CHANNEL_11	Sample state register channel 11.
ADC_SAMPLE_STATE_CHANNEL_12	Sample state register channel 12.
ADC_SAMPLE_STATE_CHANNEL_13	Sample state register channel 13.
ADC_SAMPLE_STATE_CHANNEL_14	Sample state register channel 14.
ADC_SAMPLE_STATE_CHANNEL_15	Sample state register channel 15.
ADC_SAMPLE_STATE_CHANNEL_16_TO_21	Sample state register channel 16 to 21 for unit 0 on S7G2.
ADC_SAMPLE_STATE_CHANNEL_16_TO_20	Sample state register channel 16 to 20 for unit 1 on S7G2.
ADC_SAMPLE_STATE_CHANNEL_16_TO_27	Sample state register channel 16 to 27 for unit 0 on S3A7.
ADC_SAMPLE_STATE_TEMPERATURE	Sample state register channel temperature.
ADC_SAMPLE_STATE_VOLTAGE	Sample state register channel voltage.

◆ adc_trigger_t

enum adc_trigger_t	
ADC trigger mode definitions	
Enumerator	
ADC_TRIGGER_ASYNC_EXT_TRG0	External asynchronous trigger; not for group modes.
ADC_TRIGGER_SYNC_ELC	Synchronous trigger via ELC.
ADC_TRIGGER_SOFTWARE	Software trigger; not for group modes.

◆ adc_voltage_reference_t

enum adc_voltage_reference_t	
ADC reference voltage selection (Applicable for S1/S3 series MCU's only	
Enumerator	
ADC_EXTERNAL_VOLTAGE	External voltage(VREFH0)
ADC_INTERNAL_VREF_1_5V	Internal voltage(1.5V)
ADC_INTERNAL_VREF_2_0V	Internal voltage(2.0V)
ADC_INTERNAL_VREF_2_5V	Internal voltage(2.5V)

adc_sample_state_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

adc_sample_state_reg_t	reg_id Sample state register ID.
uint8_t	num_states Number of sampling states for conversion. Ch16-20/21 use the same

	value.

Detailed Description

ADC sample state configuration

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

uint16_t	unit ADC device in use.
adc_cb_event_t	event ADC callback event.
void const *	p_context Placeholder for user data.
adc_register_t	channel Channel of conversion result. Only valid for ADC_EVENT_CONVERSION_COMPLETE.

Detailed Description

ADC callback arguments definitions

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

__I uint16_t *	p_address The address to start reading the data from.
uint32_t	length The total number of transfers to read.
transfer_size_t	transfer_size The size of each transfer.
elc_peripheral_t	elc_peripheral Name of the peripheral in the ELC list.
elc_event_t	elc_event Name of the ELC event for the peripheral.
uint32_t	calibration_data
int16_t	slope_microvolts Temperature sensor slope in microvolts/°C.
bool	calibration_ongoing Calibration is in progress.

Detailed Description

ADC Information Structure for Transfer Interface

Field Documentation

◆ calibration_data

uint32_t adc_info_t::calibration_data
Temperature sensor calibration data (0xFFFFFFFF if unsupported). Refer to hardware manual for steps on using slope with calibration data to determine temperature

---

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_channel_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

uint32_t	scan_mask
uint32_t	scan_mask_group_b Valid for group modes. Use #define ADC_MASK_CHANNEL_x from r_adc.h.
adc_group_a_t	priority_group_a Valid for group modes.
uint32_t	add_mask Valid if add enabled in Open(). Use #define ADC_MASK_CHANNEL_x from r_adc.h.
uint8_t	sample_hold_mask Channels/bits 0-2. Use #define ADC_MASK_CHANNEL_x from r_adc.h.
uint8_t	sample_hold_states Number of states to be used for sample and hold. Affects channels

	0-2.

Detailed Description

ADC channel(s) configuration

Field Documentation

◆ scan_mask

uint32_t adc_channel_cfg_t::scan_mask
Channels/bits: bit 0 is ch0; bit 15 is ch15. Use #define ADC_MASK_CHANNEL_x from r_adc.h.

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

uint16_t	unit ADC Unit to be used.
adc_mode_t	mode ADC operation mode.
adc_resolution_t	resolution ADC resolution 8, 10, or 12-bit.
adc_alignment_t	alignment Specify left or right alignment; ignored if addition used.

adc_add_t	add_average_count Add or average samples.
adc_clear_t	clearing Clear after read.
adc_trigger_t	trigger Default and Group A trigger source.
adc_trigger_t	trigger_group_b Group B trigger source; valid only for group mode.
uint8_t	scan_end_ipl Scan end interrupt priority.
uint8_t	scan_end_b_ipl Scan end group B interrupt priority.
bool	calib_adc_skip Option to perform calibration when channels are configured.
void(*	p_callback)(adc_callback_args_t *p_args) Callback function; set to NULL for none.
void const *	p_context Placeholder for user data. Passed to the user callback in adc_api_t::adc_callback_args_t.
void const *	p_extend Extension parameter for hardware specific settings.
adc_voltage_reference_t	voltage_ref ADC reference voltage selection. Default is VREF.

adc_over_current_t	over_current ADC reference voltage selection. Default is Over current.
adc_pga_t	pga0 PGA0 setting.
adc_pga_t	pga1 PGA1 setting.
adc_pga_t	pga2 PGA2 setting.

Detailed Description

ADC general configuration

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

ssp_err_t(*	open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
ssp_err_t(*	scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
ssp_err_t(*	scanStart)(adc_ctrl_t *const p_ctrl)
ssp_err_t(*	scanStop)(adc_ctrl_t *const p_ctrl)
ssp_err_t(*	scanStatusGet)(adc_ctrl_t *const p_ctrl)

ssp_err_t(*	read)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint16_t *const p_data)
ssp_err_t(*	read32)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint32_t *const p_data)
ssp_err_t(*	sampleStateCountSet)(adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
ssp_err_t(*	calibrate)(adc_ctrl_t *const p_ctrl, void *const p_extend)
ssp_err_t(*	offsetSet)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, int32_t const offset)
ssp_err_t(*	close)(adc_ctrl_t *const p_ctrl)
ssp_err_t(*	infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

ADC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ calibrate

ssp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void *const p_extend)

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.

Implemented as

- R_SDADC_Calibrate()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	Pointer to implementation specific arguments

◆ close

ssp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)

Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Implemented as

- R_ADC_Close()
- R_SDADC_Close()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ infoGet

ssp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.

Implemented as

- R_ADC_InfoGet()
- R_SDADC_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_adc_info	Pointer to ADC information structure

◆ offsetSet

ssp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, int32_t const offset)

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

Implemented as

- R_SDADC_OffsetSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_register_t)
[in]	offset	See implementation for details.

◆ open

ssp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.

Implemented as

- R_ADC_Open()
- R_SDADC_Open()

Precondition

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_cfg	Pointer to configuration structure

◆ read

ssp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint16_t *const p_data)

Read ADC conversion result.

Implemented as

- R_ADC_Read()
- R_SDADC_Read()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_register_t)
[in]	p_data	Pointer to variable to load value into.

◆ read32

ssp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint32_t *const p_data)

Read ADC conversion result into a 32-bit word.

Implemented as

- R_SDADC_Read32()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_register_t)
[in]	p_data	Pointer to variable to load value into.

◆ sampleStateCountSet

ssp_err_t(* adc_api_t::sampleStateCountSet) (adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)

Set the sample state count for the specified channel. Not supported for all implementations. See implementation for details.

Implemented as

- R_ADC_SetSampleStateCount()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_sample	Pointer to the ADC channels and corresponding sample states to be set

◆ scanCfg

ssp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)

Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.

Implemented as

- R_ADC_ScanConfigure()
- R_SDADC_ScanConfigure()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_channel_cfg	Pointer to scan configuration structure

◆ scanStart

ssp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)

Start the scan (in case of a software trigger), or enable the hardware trigger.

Implemented as

- R_ADC_ScanStart()
- R_SDADC_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanStatusGet

ssp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl)

Check scan status.

Implemented as

- R_ADC_CheckScanDone()
- R_SDADC_CheckScanDone()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanStop

ssp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

Implemented as

- R_ADC_ScanStop()
- R_SDADC_ScanStop()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ versionGet

ssp_err_t(* adc_api_t::versionGet) (ssp_version_t *const p_version)

Retrieve the API version.

Implemented as

- R_ADC_VersionGet()
- R_SDADC_VersionGet()

Precondition

This function retrieves the API version.

Parameters

[in]	p_version	Pointer to version structure

The documentation for this struct was generated from the following file:

- r_adc_api.h

adc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » ADC Interface

#include <r_adc_api.h>

Data Fields

adc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
adc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
adc_channel_cfg_t const *	p_channel_cfg Pointer to the channel configuration structure for this instance.
adc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_adc_api.h

5.1.4.2 Analog Connect Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for analog connections. More...

Data Structures

struct	analog_connect_cfg_t
struct	analog_connect_table_t
struct	analog_connect_api_t
struct	analog_connect_instance_t

Macros

#define	ANALOG_CONNECT_API_VERSION_MAJOR (2U)

Detailed Description

Interface for analog connections.

Summary

The analog connection interface allows the user to configure internal analog connections.

Implemented by: Analog Connections

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Analog Connect Interface description: Analog Connection Driver on r_analog_connect

Macro Definition Documentation

◆ ANALOG_CONNECT_API_VERSION_MAJOR

#define ANALOG_CONNECT_API_VERSION_MAJOR (2U)
Includes board and MCU related header files. Version Number of API.

analog_connect_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Analog Connect Interface

#include <r_analog_connect_api.h>

Data Fields

uint8_t	unused Placeholder for future use.

Detailed Description

User configuration structure, used in init function

The documentation for this struct was generated from the following file:

- r_analog_connect_api.h

analog_connect_table_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Analog Connect Interface

#include <r_analog_connect_api.h>

Data Fields

uint32_t	number_of_connections Number of connections in the table.
analog_connect_t const *	p_connection_table List of connections.

Detailed Description

Table of connections.

The documentation for this struct was generated from the following file:

- r_analog_connect_api.h

analog_connect_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Analog Connect Interface

#include <r_analog_connect_api.h>

Data Fields

ssp_err_t(*	init)(analog_connect_cfg_t const *const p_cfg)
ssp_err_t(*	connect)(analog_connect_t const connection)
ssp_err_t(*	connectMultiple)(analog_connect_table_t const *const p_table)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Comparator functions implemented at the HAL layer will follow this API.

Field Documentation

◆ connect

ssp_err_t(* analog_connect_api_t::connect) (analog_connect_t const connection)

Make one internal analog connection.

Implemented as

- R_ANALOG_CONNECT_Connect()

Parameters

[in]	connection	Internal analog connection to make

◆ connectMultiple

ssp_err_t(* analog_connect_api_t::connectMultiple) (analog_connect_table_t const *const p_table)

Make multiple internal analog connections. Connections are made in the order they are listed in the table. This API is most efficient when all connections for the same module/channel combination are grouped together.

Implemented as

- R_ANALOG_CONNECT_ConnectMultiple()

Parameters

[in]	p_table	Pointer to table of internal analog connection to make

◆ init

ssp_err_t(* analog_connect_api_t::init) (analog_connect_cfg_t const *const p_cfg)

Initialize the analog connect module.

Implemented as

- R_ANALOG_CONNECT_Init()

Parameters

[in]	p_cfg	Pointer to configuration

◆ versionGet

ssp_err_t(* analog_connect_api_t::versionGet) (ssp_version_t *const p_version)

Retrieve the API version.

Implemented as

- R_ANALOG_CONNECT_VersionGet()

Parameters

[in]	p_version	Pointer to version structure

The documentation for this struct was generated from the following file:

- r_analog_connect_api.h

analog_connect_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Analog Connect Interface

#include <r_analog_connect_api.h>

Data Fields

analog_connect_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
analog_connect_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_analog_connect_api.h

5.1.4.3 CAC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for clock frequency accuracy measurements. More...

Data Structures

struct	cac_ref_clock_config_t
struct	cac_meas_clock_config_t
struct	cac_callback_args_t
struct	cac_cfg_t

struct	cac_api_t
struct	cac_instance_t

Typedefs

typedef void	cac_ctrl_t

Enumerations

enum	cac_event_t { CAC_EVENT_FREQUENCY_ERROR, CAC_EVENT_MEASUREMENT_COMPLETE, CAC_EVENT_COUNTER_OVERFLOW }
enum	cac_clock_type_t { CAC_CLOCK_MEASURED, CAC_CLOCK_REFERENCE }
enum	cac_clock_source_t { CAC_CLOCK_SOURCE_MAIN_OSC = 0x00, CAC_CLOCK_SOURCE_SUBCLOCK = 0x01, CAC_CLOCK_SOURCE_HOCO = 0x02, CAC_CLOCK_SOURCE_MOCO = 0x03, CAC_CLOCK_SOURCE_LOCO = 0x04, CAC_CLOCK_SOURCE_PCLKB = 0x05, CAC_CLOCK_SOURCE_IWDT = 0x06, CAC_CLOCK_SOURCE_MEAS_MAX = CAC_CLOCK_SOURCE_IWDT, CAC_CLOCK_SOURCE_EXTERNAL = 0x07, CAC_CLOCK_SOURCE_REF_MAX = CAC_CLOCK_SOURCE_EXTERNAL }
enum	cac_ref_divider_t { CAC_REF_DIV_32 = 0x00, CAC_REF_DIV_128 = 0x01, CAC_REF_DIV_1024 = 0x02, CAC_REF_DIV_8192 = 0x03 }
enum	cac_ref_digfilter_t { CAC_REF_DIGITAL_FILTER_OFF = 0x00, CAC_REF_DIGITAL_FILTER_1 = 0x01, CAC_REF_DIGITAL_FILTER_4 = 0x02, CAC_REF_DIGITAL_FILTER_16 = 0x03 }
enum	cac_ref_edge_t { CAC_REF_EDGE_RISE = 0x00, CAC_REF_EDGE_FALL = 0x01, CAC_REF_EDGE_BOTH = 0x02 }
enum	cac_meas_divider_t { CAC_MEAS_DIV_1 = 0x00, CAC_MEAS_DIV_4 = 0x01, CAC_MEAS_DIV_8 = 0x02, CAC_MEAS_DIV_32 = 0x03 }

Detailed Description

Interface for clock frequency accuracy measurements.

Register definitions, common services and error codes.

Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to

check a system clock frequency with a reference clock signal by counting the number of pulses of the clock (system clock) to be measured.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CAC Interface description: Clock Accurate Circuit Driver

Typedef Documentation

◆ cac_ctrl_t

typedef void cac_ctrl_t
CAC control block. Allocate an instance specific control block to pass into the CAC API calls. **Implemented as** ◦ cac_instance_ctrl_t

Enumeration Type Documentation

◆ cac_clock_source_t

enum cac_clock_source_t	
Enumeration of the possible clock sources for both the reference and measurement clocks.	
Enumerator	
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_LOCO	LOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_IWDT	IWDT- Dedicated on-chip oscillator.
CAC_CLOCK_SOURCE_MEAS_MAX	Maximum measured clock.
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.
CAC_CLOCK_SOURCE_REF_MAX	Maximum reference clock.

◆ cac_clock_type_t

enum cac_clock_type_t	
Enumeration of the two possible clocks.	
Enumerator	
CAC_CLOCK_MEASURED	Measurement clock.
CAC_CLOCK_REFERENCE	Reference clock.

◆ cac_event_t

enum cac_event_t	
Event types returned by the ISR callback when used in CAC interrupt mode	
Enumerator	
CAC_EVENT_FREQUENCY_ERROR	Frequency error.
CAC_EVENT_MEASUREMENT_COMPLETE	Measurement complete.
CAC_EVENT_COUNTER_OVERFLOW	Counter overflow.

◆ cac_meas_divider_t

enum cac_meas_divider_t	
Enumeration of available dividers for the measurement clock	
Enumerator	
CAC_MEAS_DIV_1	Measurement clock divided by 1.
CAC_MEAS_DIV_4	Measurement clock divided by 4.
CAC_MEAS_DIV_8	Measurement clock divided by 8.
CAC_MEAS_DIV_32	Measurement clock divided by 32.

◆ cac_ref_digfilter_t

enum cac_ref_digfilter_t	
Enumeration of available digital filter settings for an external reference clock.	
Enumerator	
CAC_REF_DIGITAL_FILTER_OFF	No digital filter on the CACREF pin for reference clock.
CAC_REF_DIGITAL_FILTER_1	Sampling clock for digital filter = measuring frequency.
CAC_REF_DIGITAL_FILTER_4	Sampling clock for digital filter = measuring frequency/4.
CAC_REF_DIGITAL_FILTER_16	Sampling clock for digital filter = measuring frequency/16.

◆ cac_ref_divider_t

enum cac_ref_divider_t	
Enumeration of available dividers for the reference clock.	
Enumerator	
CAC_REF_DIV_32	Reference clock divided by 32.
CAC_REF_DIV_128	Reference clock divided by 128.
CAC_REF_DIV_1024	Reference clock divided by 1024.
CAC_REF_DIV_8192	Reference clock divided by 8192.

◆ cac_ref_edge_t

enum cac_ref_edge_t	
Enumeration of available edge detect settings for the reference clock.	
Enumerator	
CAC_REF_EDGE_RISE	Rising edge detect for the Reference clock.
CAC_REF_EDGE_FALL	Falling edge detect for the Reference clock.
CAC_REF_EDGE_BOTH	Both Rising and Falling edges detect for the Reference clock.

cac_ref_clock_config_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

cac_ref_divider_t	divider Divider specification for the Reference clock.
cac_clock_source_t	clock Clock source for the Reference clock.

cac_ref_digfilter_t	digfilter Digital filter selection for the CACREF ext clock.
cac_ref_edge_t	edge Edge detection for the Reference clock.

Detailed Description

Structure defining the settings that apply to reference clock configuration.

The documentation for this struct was generated from the following file:

- r_cac_api.h

cac_meas_clock_config_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

cac_meas_divider_t	divider Divider specification for the Measurement clock.
cac_clock_source_t	clock Clock source for the Measurement clock.

Detailed Description

Structure defining the settings that apply to measurement clock configuration.

The documentation for this struct was generated from the following file:

- r_cac_api.h

cac_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

cac_event_t	event The event can be used to identify what caused the callback (cac ready or error).
void const *	p_context Placeholder for user data. Set in cac_api_t::open function in cac_cfg_t .

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_cac_api.h

cac_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

cac_ref_clock_config_t	cac_ref_clock reference clock specific settings
cac_meas_clock_config_t	cac_meas_clock measurement clock specific settings

uint16_t	cac_upper_limit the upper limit counter threshold
uint16_t	cac_lower_limit the lower limit counter threshold
bool	mei_interrupt_enabled True if Measurement Complete interrupt is enabled.
bool	ovf_interrupt_enabled True if Overflow interrupt is enabled.
bool	ferr_interrupt_enabled True if Frequency Error interrupt is enabled.
bool	continuous_mode True if measurement continuously restarts after completing.
uint8_t	frequency_error_ipl Frequency error interrupt priority.
uint8_t	measurement_end_ipl Measurement end interrupt priority.
uint8_t	overflow_ipl Overflow interrupt priority.
void(*	p_callback)(cac_callback_args_t *p_args) Callback provided when a CAC interrupt ISR occurs.
void const *	p_extend CAC hardware dependent configuration.
void const *	p_context

	Placeholder for user data. Passed to user callback in cac_callback_args_t.

Detailed Description

CAC Configuration

The documentation for this struct was generated from the following file:

- r_cac_api.h

cac_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

ssp_err_t(*	open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
ssp_err_t(*	read)(cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)
ssp_err_t(*	close)(cac_ctrl_t *const p_ctrl)
ssp_err_t(*	stopMeasurement)(cac_ctrl_t *const p_ctrl)
ssp_err_t(*	startMeasurement)(cac_ctrl_t *const p_ctrl)
ssp_err_t(*	reset)(cac_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

CAC functions implemented at the HAL layer API

Field Documentation

◆ close

ssp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)

Close function for CAC device.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ open

ssp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

Parameters

[out]	p_ctrl	Pointer to CAC device control. Must be declared by user. Value set here.
[in]	cac_cfg_t	Pointer to CAC configuration structure. All elements of this structure must be set by user.

◆ read

ssp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)

Read function for CAC peripheral.

Parameters

[in]	p_ctrl	Control for the CAC device context.
[in]	p_status	Pointer to variable in which to store the current CASTR register contents.
[in]	p_counter	Pointer to variable in which to store the current CACNTBR register contents.

◆ reset

ssp_err_t(* cac_api_t::reset) (cac_ctrl_t *const p_ctrl)

Reset function for CAC device.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ startMeasurement

ssp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ stopMeasurement

ssp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)

End a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ versionGet

ssp_err_t(* cac_api_t::versionGet) (ssp_version_t *p_version)

Get the CAC API and code version information.

Parameters

[out]	p_version	is value returned.

The documentation for this struct was generated from the following file:

- r_cac_api.h

cac_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAC Interface

#include <r_cac_api.h>

Data Fields

cac_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
cac_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
cac_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_cac_api.h

5.1.4.4 CAN Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for CAN peripheral. More...

Data Structures

union	can_status_t
union	can_error_t

struct	can_callback_args_t
struct	can_bit_timing_cfg_t
struct	can_frame_t
struct	can_mailbox_t
struct	can_cfg_t
struct	can_api_t
struct	can_instance_t

Typedefs

typedef uint32_t	can_id_t
typedef void	can_ctrl_t

Enumerations

enum	can_event_t { CAN_EVENT_RX_COMPLETE, CAN_EVENT_TX_COMPLETE, CAN_EVENT_ERR_BUS_OFF, CAN_EVENT_BUS_RECOVERY, CAN_EVENT_ERR_PASSIVE, CAN_EVENT_ERR_WARNING, CAN_EVENT_MAILBOX_OVERWRITE_OVERRUN = 6, CAN_EVENT_MAILBOX_OVERRUN = 6 }
enum	can_mode_t { CAN_MODE_NORMAL, CAN_MODE_HALT, CAN_MODE_SLEEP, CAN_MODE_EXIT_SLEEP, CAN_MODE_RESET, CAN_MODE_LISTEN, CAN_MODE_LOOPBACK_INTERNAL, CAN_MODE_LOOPBACK_EXTERNAL }
enum	can_id_mode_t { CAN_ID_MODE_STANDARD, CAN_ID_MODE_EXTENDED }
enum	can_frame_type_t { CAN_FRAME_TYPE_DATA, CAN_FRAME_TYPE_REMOTE }
enum	can_message_mode_t { CAN_MESSAGE_MODE_OVERWRITE, CAN_MESSAGE_MODE_OVERRUN }
enum	can_clock_source_t { CAN_CLOCK_SOURCE_PCLKB, CAN_CLOCK_SOURCE_CANMCLK }
enum	can_time_segment1_t {

	CAN_TIME_SEGMENT1_TQ4 = 3, CAN_TIME_SEGMENT1_TQ5, CAN_TIME_SEGMENT1_TQ6, CAN_TIME_SEGMENT1_TQ7, CAN_TIME_SEGMENT1_TQ8, CAN_TIME_SEGMENT1_TQ9, CAN_TIME_SEGMENT1_TQ10, CAN_TIME_SEGMENT1_TQ11, CAN_TIME_SEGMENT1_TQ12, CAN_TIME_SEGMENT1_TQ13, CAN_TIME_SEGMENT1_TQ14, CAN_TIME_SEGMENT1_TQ15, CAN_TIME_SEGMENT1_TQ16 }
enum	can_time_segment2_t { CAN_TIME_SEGMENT2_TQ2 = 1, CAN_TIME_SEGMENT2_TQ3, CAN_TIME_SEGMENT2_TQ4, CAN_TIME_SEGMENT2_TQ5, CAN_TIME_SEGMENT2_TQ6, CAN_TIME_SEGMENT2_TQ7, CAN_TIME_SEGMENT2_TQ8 }
enum	can_sync_jump_width_t { CAN_SYNC_JUMP_WIDTH_TQ1 = 0, CAN_SYNC_JUMP_WIDTH_TQ2, CAN_SYNC_JUMP_WIDTH_TQ3, CAN_SYNC_JUMP_WIDTH_TQ4 }
enum	can_mailbox_send_receive_t { CAN_MAILBOX_RECEIVE, CAN_MAILBOX_TRANSMIT }
enum	can_command_t { CAN_COMMAND_MODE_SWITCH = 1 }

Detailed Description

Interface for CAN peripheral.

Summary

The CAN interface provides common APIs for CAN HAL drivers. CAN interface supports following features.

- Full-duplex CAN communication
- Generic CAN parameter setting
- Interrupt driven transmit/receive processing
- Callback function support with returning event code
- Hardware resource locking during a transaction

Typedef Documentation

◆ can_ctrl_t

typedef void can_ctrl_t

CAN control block. Allocate an instance specific control block to pass into the CAN API calls.

Implemented as

- can_instance_ctrl_t

◆ can_id_t

typedef uint32_t can_id_t
CAN Id

Enumeration Type Documentation

◆ can_clock_source_t

enum can_clock_source_t	
CAN Source Clock	
Enumerator	
CAN_CLOCK_SOURCE_PCLKB	PCLB is the source of the CAN Clock.
CAN_CLOCK_SOURCE_CANMCLK	CANMCLK is the source of the CAN Clock.

◆ can_command_t

enum can_command_t	
CAN control commands.	
Enumerator	
CAN_COMMAND_MODE_SWITCH	Switch CAN operating mode..

◆ can_event_t

enum can_event_t	
CAN event codes	
Enumerator	
CAN_EVENT_RX_COMPLETE	Receive complete event.
CAN_EVENT_TX_COMPLETE	Transmit complete event.
CAN_EVENT_ERR_BUS_OFF	Bus Off event.
CAN_EVENT_BUS_RECOVERY	Bus Off Recovery event.
CAN_EVENT_ERR_PASSIVE	Error Passive event.
CAN_EVENT_ERR_WARNING	Error Warning event.
CAN_EVENT_MAILBOX_OVERWRITE_OVERRUN	DEPRECATED, Mailbox has been overrun. This event is not used when the mailbox is overwritten.
CAN_EVENT_MAILBOX_OVERRUN	Mailbox has been overrun.

◆ can_frame_type_t

enum can_frame_type_t	
CAN frame types	
Enumerator	
CAN_FRAME_TYPE_DATA	Data frame type.
CAN_FRAME_TYPE_REMOTE	Remote frame type.

◆ can_id_mode_t

enum can_id_mode_t	
CAN ID modes	
Enumerator	
CAN_ID_MODE_STANDARD	Standard IDs of 11 bits used.
CAN_ID_MODE_EXTENDED	Extended IDs of 29 bits used.

◆ can_mailbox_send_receive_t

enum can_mailbox_send_receive_t	
CAN Mailbox type	
Enumerator	
CAN_MAILBOX_RECEIVE	Mailbox is for receiving.
CAN_MAILBOX_TRANSMIT	Mailbox is for sending.

◆ can_message_mode_t

enum can_message_mode_t	
CAN Message Modes	
Enumerator	
CAN_MESSAGE_MODE_OVERWRITE	Receive data will be overwritten if not read before the next frame.
CAN_MESSAGE_MODE_OVERRUN	Receive data will be retained until it is read.

◆ can_mode_t

enum can_mode_t	
CAN Operation modes	
Enumerator	
CAN_MODE_NORMAL	CAN Normal Mode.
CAN_MODE_HALT	CAN Halt Mode.
CAN_MODE_SLEEP	CAN SLEEP Mode.
CAN_MODE_EXIT_SLEEP	CAN Exit SLEEP Mode.
CAN_MODE_RESET	CAN Reset Mode.
CAN_MODE_LISTEN	CAN Listen Mode.
CAN_MODE_LOOPBACK_INTERNAL	CAN Internal Loopback Mode.
CAN_MODE_LOOPBACK_EXTERNAL	CAN External Loopback Mode.

◆ can_sync_jump_width_t

enum can_sync_jump_width_t	
CAN Synchronization Jump Width Time Quanta	
Enumerator	
CAN_SYNC_JUMP_WIDTH_TQ1	Synchronization Jump Width setting for 1 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ2	Synchronization Jump Width setting for 2 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ3	Synchronization Jump Width setting for 3 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ4	Synchronization Jump Width setting for 4 Time Quanta.

◆ can_time_segment1_t

enum can_time_segment1_t	
CAN Time Segment 1 Time Quanta	
Enumerator	
CAN_TIME_SEGMENT1_TQ4	Time Segment 1 setting for 4 Time Quanta.
CAN_TIME_SEGMENT1_TQ5	Time Segment 1 setting for 5 Time Quanta.
CAN_TIME_SEGMENT1_TQ6	Time Segment 1 setting for 6 Time Quanta.
CAN_TIME_SEGMENT1_TQ7	Time Segment 1 setting for 7 Time Quanta.
CAN_TIME_SEGMENT1_TQ8	Time Segment 1 setting for 8 Time Quanta.
CAN_TIME_SEGMENT1_TQ9	Time Segment 1 setting for 9 Time Quanta.
CAN_TIME_SEGMENT1_TQ10	Time Segment 1 setting for 10 Time Quanta.
CAN_TIME_SEGMENT1_TQ11	Time Segment 1 setting for 11 Time Quanta.
CAN_TIME_SEGMENT1_TQ12	Time Segment 1 setting for 12 Time Quanta.
CAN_TIME_SEGMENT1_TQ13	Time Segment 1 setting for 13 Time Quanta.
CAN_TIME_SEGMENT1_TQ14	Time Segment 1 setting for 14 Time Quanta.
CAN_TIME_SEGMENT1_TQ15	Time Segment 1 setting for 15 Time Quanta.
CAN_TIME_SEGMENT1_TQ16	Time Segment 1 setting for 16 Time Quanta.

◆ can_time_segment2_t

enum can_time_segment2_t	
CAN Time Segment 2 Time Quanta	
Enumerator	
CAN_TIME_SEGMENT2_TQ2	Time Segment 2 setting for 2 Time Quanta.
CAN_TIME_SEGMENT2_TQ3	Time Segment 2 setting for 3 Time Quanta.
CAN_TIME_SEGMENT2_TQ4	Time Segment 2 setting for 4 Time Quanta.
CAN_TIME_SEGMENT2_TQ5	Time Segment 2 setting for 5 Time Quanta.
CAN_TIME_SEGMENT2_TQ6	Time Segment 2 setting for 6 Time Quanta.
CAN_TIME_SEGMENT2_TQ7	Time Segment 2 setting for 7 Time Quanta.
CAN_TIME_SEGMENT2_TQ8	Time Segment 2 setting for 8 Time Quanta.

can_status_t Union Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Detailed Description

CAN Status

The documentation for this union was generated from the following file:

- r_can_api.h

can_error_t Union Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Detailed Description

CAN Error Code

The documentation for this union was generated from the following file:

- r_can_api.h

can_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

uint32_t	channel Device channel number.
can_event_t	event Event code.
uint32_t	mailbox Mailbox number of interrupt source.
void const *	p_context Context provided to user during callback.

Detailed Description

CAN callback parameter definition

The documentation for this struct was generated from the following file:

- r_can_api.h

can_bit_timing_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

uint32_t	baud_rate_prescaler Baud rate prescaler. Valid values: 1 - 1024.
can_time_segment1_t	time_segment_1 Time segment 1 control.
can_time_segment2_t	time_segment_2 Time segment 2 control.
can_sync_jump_width_t	synchronization_jump_width Synchronization jump width.

Detailed Description

CAN bit rate configuration.

The documentation for this struct was generated from the following file:

- r_can_api.h

can_frame_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

can_id_t	id CAN id.

uint8_t	data_length_code CAN Data Length code, number of bytes in the message.
uint8_t	data [8] CAN data, up to 8 bytes.
can_frame_type_t	type Frame type, data or remote frame.

Detailed Description

CAN data Frame

The documentation for this struct was generated from the following file:

- r_can_api.h

can_mailbox_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

can_id_t	mailbox_id Mailbox ID.
can_mailbox_send_receive_t	mailbox_type Receive or Transmit mailbox type.
can_frame_type_t	frame_type Frame type for receive mailbox.

Detailed Description

CAN Mailbox

The documentation for this struct was generated from the following file:

- r_can_api.h

can_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

uint32_t	channel CAN channel.
can_bit_timing_cfg_t *	p_bit_timing CAN bit timing.
can_id_mode_t	id_mode Standard or Extended ID mode.
uint32_t	mailbox_count Number of mailboxes.
can_mailbox_t *	p_mailbox Pointer to mailboxes.
can_message_mode_t	message_mode Overwrite message or overrun.
void(*	p_callback)(can_callback_args_t *p_args) Pointer to callback function.

void const *	p_context User defined callback context.
void const *	p_extend CAN hardware dependent configuration.
uint8_t	error_ipl Error interrupt priority.
uint8_t	mailbox_rx_ipl Receive interrupt priority.
uint8_t	mailbox_tx_ipl Transmit interrupt priority.

Detailed Description

CAN Configuration

The documentation for this struct was generated from the following file:

- r_can_api.h

can_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

ssp_err_t(*	open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
ssp_err_t(*	read)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
ssp_err_t(*	write)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

ssp_err_t(*	close)(can_ctrl_t *const p_ctrl)
ssp_err_t(*	control)(can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)
ssp_err_t(*	infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Shared Interface definition for CAN

Field Documentation

◆ close

ssp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)

Close function for CAN device

Implemented as

- R_CAN_Close()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.

◆ control

ssp_err_t(* can_api_t::control) (can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)

Control function for CAN device

Implemented as

- R_CAN_Control()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	command	Command type.
[in]	p_data	Command data.

◆ infoGet

ssp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)

Get CAN channel info.

Implemented as

- R_CAN_InfoGet()

Parameters

[in]	p_ctrl	Handle for channel (pointer to channel control block)
[out]	p_info	Memory address to return channel specific data to.

◆ open

ssp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

Open function for CAN device

Implemented as

- R_CAN_Open()

Parameters

[in,out]	p_ctrl	Pointer to the CAN control block Must be declared by user. Value set here.
[in]	can_cfg_t	Pointer to CAN configuration structure. All elements of this structure must be set by user.

◆ read

ssp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

Read function for CAN device, non-Blocking.

Implemented as

- R_CAN_Read()

Parameters

[in]	p_ctrl	Pointer to the CAN control block for the channel.
[in]	mailbox	Mailbox to read from.
[out]	p_frame	Pointer for frame of CAN ID, DLC, data and frame type.

◆ versionGet

ssp_err_t(* can_api_t::versionGet) (ssp_version_t *const p_version)

Version get function for CAN device

Implemented as

- R_CAN_VersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the version information

◆ write

ssp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

Write function for CAN device

Implemented as

- R_CAN_Write()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	mailbox	Mailbox to write to.
[in]	p_frame	Pointer for frame of CAN ID, DLC, data and frame type to write.

The documentation for this struct was generated from the following file:

- r_can_api.h

can_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CAN Interface

#include <r_can_api.h>

Data Fields

can_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
can_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
can_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_can_api.h

5.1.4.5 CGC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for clock generation. More...

Data Structures

struct	cgc_callback_args_t
struct	cgc_clock_cfg_t
struct	cgc_system_clock_cfg_t
struct	cgc_clocks_cfg_t
struct	cgc_api_t
struct	cgc_instance_t

Enumerations

enum	cgc_event_t { CGC_EVENT_OSC_STOP_DETECT }
enum	cgc_clock_t { CGC_CLOCK_HOCO = 0x00, CGC_CLOCK_MOCO = 0x01, CGC_CLOCK_LOCO = 0x02, CGC_CLOCK_MAIN_OSC = 0x03, CGC_CLOCK_SUBCLOCK = 0x04, CGC_CLOCK_PLL = 0x05 }
enum	cgc_pll_div_t { CGC_PLL_DIV_1 = 0x00, CGC_PLL_DIV_2 = 0x01, CGC_PLL_DIV_3 = 0x02, CGC_PLL_DIV_4 = 0x03 }
enum	cgc_sys_clock_div_t { CGC_SYS_CLOCK_DIV_1 = 0x00, CGC_SYS_CLOCK_DIV_2 = 0x01, CGC_SYS_CLOCK_DIV_4 = 0x02, CGC_SYS_CLOCK_DIV_8 = 0x03, CGC_SYS_CLOCK_DIV_16 = 0x04, CGC_SYS_CLOCK_DIV_32 = 0x05, CGC_SYS_CLOCK_DIV_64 = 0x06 }

enum	cgc_system_clocks_t { CGC_SYSTEM_CLOCKS_PCLKA, CGC_SYSTEM_CLOCKS_PCLKB, CGC_SYSTEM_CLOCKS_PCLKC, CGC_SYSTEM_CLOCKS_PCLKD, CGC_SYSTEM_CLOCKS_BCLK, CGC_SYSTEM_CLOCKS_FCLK, CGC_SYSTEM_CLOCKS_ICLK }
enum	cgc_clockout_dividers_t { CGC_CLOCKOUT_DIV_1 = 0x00, CGC_CLOCKOUT_DIV_2 = 0x01, CGC_CLOCKOUT_DIV_4 = 0x02, CGC_CLOCKOUT_DIV_8 = 0x03, CGC_CLOCKOUT_DIV_16 = 0x04, CGC_CLOCKOUT_DIV_32 = 0x05, CGC_CLOCKOUT_DIV_64 = 0x06, CGC_CLOCKOUT_DIV_128 = 0x07 }
enum	cgc_bclockout_dividers_t { CGC_BCLOCKOUT_DIV_1 = 0x00, CGC_BCLOCKOUT_DIV_2 = 0x01 }
enum	cgc_usb_clock_div_t { CGC_USB_CLOCK_DIV_3 = 0x02, CGC_USB_CLOCK_DIV_4 = 0x03, CGC_USB_CLOCK_DIV_5 = 0x04 }
enum	cgc_systick_period_units_t { CGC_SYSTICK_PERIOD_UNITS_MILLISECONDS = 1000, CGC_SYSTICK_PERIOD_UNITS_MICROSECONDS = 1000000 }
enum	cgc_clock_change_t { CGC_CLOCK_CHANGE_NONE, CGC_CLOCK_CHANGE_STOP, CGC_CLOCK_CHANGE_START }

Detailed Description

Interface for clock generation.

Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CGC Interface description: CGC Driver

Enumeration Type Documentation

◆ cgc_bclockout_dividers_t

enum cgc_bclockout_dividers_t	
Divider values for the external bus clock output.	
Enumerator	
CGC_BCLOCKOUT_DIV_1	External bus clock source is divided by 1.
CGC_BCLOCKOUT_DIV_2	External bus clock source is divided by 2.

◆ cgc_clock_change_t

enum cgc_clock_change_t	
Clock options	
Enumerator	
CGC_CLOCK_CHANGE_NONE	No change to the clock.
CGC_CLOCK_CHANGE_STOP	Stop the clock.
CGC_CLOCK_CHANGE_START	Start the clock.

◆ cgc_clock_t

enum cgc_clock_t	
System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider	
Enumerator	
CGC_CLOCK_HOCO	The high speed on chip oscillator.
CGC_CLOCK_MOCO	The middle speed on chip oscillator.
CGC_CLOCK_LOCO	The low speed on chip oscillator.
CGC_CLOCK_MAIN_OSC	The main oscillator.
CGC_CLOCK_SUBCLOCK	The subclock oscillator.
CGC_CLOCK_PLL	The PLL oscillator.

◆ cgc_clockout_dividers_t

enum cgc_clockout_dividers_t	
Divider values for the CLKOUT output.	
Enumerator	
CGC_CLOCKOUT_DIV_1	Clockout source is divided by 1.
CGC_CLOCKOUT_DIV_2	Clockout source is divided by 2.
CGC_CLOCKOUT_DIV_4	Clockout source is divided by 4.
CGC_CLOCKOUT_DIV_8	Clockout source is divided by 8.
CGC_CLOCKOUT_DIV_16	Clockout source is divided by 16.
CGC_CLOCKOUT_DIV_32	Clockout source is divided by 32.
CGC_CLOCKOUT_DIV_64	Clockout source is divided by 64.
CGC_CLOCKOUT_DIV_128	Clockout source is divided by 128.

◆ cgc_event_t

enum cgc_event_t	
Events that can trigger a callback function	
Enumerator	
CGC_EVENT_OSC_STOP_DETECT	Oscillator stop detection has caused the event.

◆ cgc_pll_div_t

enum cgc_pll_div_t	
PLL divider values	
Enumerator	
CGC_PLL_DIV_1	PLL divider of 1.
CGC_PLL_DIV_2	PLL divider of 2.
CGC_PLL_DIV_3	PLL divider of 3 (S7G2 only).
CGC_PLL_DIV_4	PLL divider of 4 (S3A7 only).

◆ cgc_sys_clock_div_t

enum cgc_sys_clock_div_t	
System clock divider values - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D	
Enumerator	
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.

◆ cgc_system_clocks_t

enum cgc_system_clocks_t	
System clock identifiers - Used as an input parameter to the cgc_api_t::systemClockFreqGet function.	
Enumerator	
CGC_SYSTEM_CLOCKS_PCLKA	PCLKA - Peripheral module clock A.
CGC_SYSTEM_CLOCKS_PCLKB	PCLKB - Peripheral module clock B.
CGC_SYSTEM_CLOCKS_PCLKC	PCLKC - Peripheral module clock C.
CGC_SYSTEM_CLOCKS_PCLKD	PCLKD - Peripheral module clock D.
CGC_SYSTEM_CLOCKS_BCLK	BCLK - External bus Clock.
CGC_SYSTEM_CLOCKS_FCLK	FCLK - FlashIF clock.
CGC_SYSTEM_CLOCKS_ICLK	ICLK - System clock.

◆ cgc_systick_period_units_t

enum cgc_systick_period_units_t	
Available period units for R_CGC_SystickUpdate()	
Enumerator	
CGC_SYSTICK_PERIOD_UNITS_MILLISECONDS	Requested period in milliseconds.
CGC_SYSTICK_PERIOD_UNITS_MICROSECONDS	Requested period in microseconds.

◆ cgc_usb_clock_div_t

enum cgc_usb_clock_div_t	
USB clock divider values	
Enumerator	
CGC_USB_CLOCK_DIV_3	Divide USB source clock by 3.
CGC_USB_CLOCK_DIV_4	Divide USB source clock by 4.
CGC_USB_CLOCK_DIV_5	Divide USB source clock by 5.

cgc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

cgc_event_t	event The event can be used to identify what caused the callback.
void const *	p_context Placeholder for user data.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_cgc_api.h

cgc_clock_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

cgc_clock_t	source_clock PLL source clock (S7G2 only).
cgc_pll_div_t	divider PLL divider.
float	multiplier

	PLL multiplier.

Detailed Description

Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock.

The documentation for this struct was generated from the following file:

- r_cgc_api.h

cgc_system_clock_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

cgc_sys_clock_div_t	pclka_div Divider value for PCLKA.
cgc_sys_clock_div_t	pclkb_div Divider value for PCLKB.
cgc_sys_clock_div_t	pclkc_div Divider value for PCLKC.
cgc_sys_clock_div_t	pclkd_div Divider value for PCLKD.
cgc_sys_clock_div_t	bclk_div Divider value for BCLK.
cgc_sys_clock_div_t	fclk_div Divider value for FCLK.

cgc_sys_clock_div_t	iclk_div Divider value for ICLK.

Detailed Description

Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions.

The documentation for this struct was generated from the following file:

- r_cgc_api.h

cgc_clocks_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

cgc_clock_t	system_clock System clock source enumeration.
cgc_clock_cfg_t	pll_cfg PLL configuration structure.
cgc_system_clock_cfg_t	sys_cfg Clock dividers structure.
cgc_clock_change_t	loco_state State of LOCO.
cgc_clock_change_t	moco_state State of MOCO.
cgc_clock_change_t	hoco_state

	State of HOCO.
cgc_clock_change_t	subosc_state State of Sub-oscillator.
cgc_clock_change_t	mainosc_state State of Main oscillator.
cgc_clock_change_t	pll_state State of PLL.

Detailed Description

Clock configuration

The documentation for this struct was generated from the following file:

- r_cgc_api.h

cgc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

ssp_err_t(*	init)(void)
ssp_err_t(*	clocksCfg)(cgc_clocks_cfg_t const *const p_clock_cfg)
ssp_err_t(*	clockStart)(cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg)
ssp_err_t(*	clockStop)(cgc_clock_t clock_source)
ssp_err_t(*	systemClockSet)(cgc_clock_t clock_source, cgc_system_clock_cfg_t const *const p_clock_cfg)
ssp_err_t(*	systemClockGet)(cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)

ssp_err_t(*	systemClockFreqGet)(cgc_system_clocks_t clock, uint32_t *p_freq_hz)
ssp_err_t(*	clockCheck)(cgc_clock_t clock_source)
ssp_err_t(*	oscStopDetect)(void(*p_callback)(cgc_callback_args_t *p_args), bool enable)
ssp_err_t(*	oscStopStatusClear)(void)
ssp_err_t(*	busClockOutCfg)(cgc_bclockout_dividers_t divider)
ssp_err_t(*	busClockOutEnable)(void)
ssp_err_t(*	busClockOutDisable)(void)
ssp_err_t(*	clockOutCfg)(cgc_clock_t clock, cgc_clockout_dividers_t divider)
ssp_err_t(*	clockOutEnable)(void)
ssp_err_t(*	clockOutDisable)(void)
ssp_err_t(*	lcdClockCfg)(cgc_clock_t clock)
ssp_err_t(*	lcdClockEnable)(void)
ssp_err_t(*	lcdClockDisable)(void)
ssp_err_t(*	sdadcClockCfg)(cgc_clock_t clock)
ssp_err_t(*	sdadcClockEnable)(void)
ssp_err_t(*	sdadcClockDisable)(void)
ssp_err_t(*	sdramClockOutEnable)(void)
ssp_err_t(*	sdramClockOutDisable)(void)
ssp_err_t(*	usbClockCfg)(cgc_usb_clock_div_t divider)
ssp_err_t(*	systickUpdate)(uint32_t period_count, cgc_systick_period_units_t units)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

CGC functions implemented at the HAL layer follow this API.

Field Documentation

◆ busClockOutCfg

ssp_err_t(* cgc_api_t::busClockOutCfg) (cgc_bclockout_dividers_t divider)

Configure the bus clock output secondary divider. The primary divider is set using the bsp clock configuration and the cgc_api_t::systemClockSet function (S7G2 and S3A7 only).

Implemented as

- R_CGC_BusClockOutCfg()

Parameters

[in]	divider	The divider of 1 or 2 of the clock source.

◆ busClockOutDisable

ssp_err_t(* cgc_api_t::busClockOutDisable) (void)

Disable the bus clock output (S7G2 and S3A7 only).

Implemented as

- R_CGC_BusClockOutDisable()

◆ busClockOutEnable

ssp_err_t(* cgc_api_t::busClockOutEnable) (void)

Enable the bus clock output (S7G2 and S3A7 only).

Implemented as

- R_CGC_BusClockOutEnable()

◆ clockCheck

ssp_err_t(* cgc_api_t::clockCheck) (cgc_clock_t clock_source)

Check the stability of the selected clock.

Implemented as

- R_CGC_ClockCheck()

Parameters

[in]	clock_source	Which clock source to check for stability.

◆ clockOutCfg

ssp_err_t(* cgc_api_t::clockOutCfg) (cgc_clock_t clock, cgc_clockout_dividers_t divider)

Configure clockOut.

Implemented as

- R_CGC_ClockOutCfg()

Parameters

[in]	clock	Clock source.
[in]	divider	Divider of between 1 and 128 of the clock source.

◆ clockOutDisable

ssp_err_t(* cgc_api_t::clockOutDisable) (void)

Disable clock output on the CLKOUT pin. The source of the clock is controlled by cgc_api_t::clockOutCfg.

Implemented as

- R_CGC_ClockOutDisable()

◆ clockOutEnable

ssp_err_t(* cgc_api_t::clockOutEnable) (void)

Enable clock output on the CLKOUT pin. The source of the clock is controlled by cgc_api_t::clockOutCfg.

Implemented as

- R_CGC_ClockOutEnable()

◆ clocksCfg

ssp_err_t(* cgc_api_t::clocksCfg) (cgc_clocks_cfg_t const *const p_clock_cfg)

Configure all system clocks.

Implemented as

- R_CGC_ClocksCfg()

Note

The BSP module calls this function at startup, but it can also be called from the application to change clocks at runtime.

Parameters

[in]	p_clock_cfg	Pointer to a structure that contains the dividers or multipliers to be used when configuring the PLL.

◆ clockStart

ssp_err_t(* cgc_api_t::clockStart) (cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg)

Start a clock.

Implemented as

- R_CGC_ClockStart()

Precondition

Clock to be started must not be running prior to calling this function or an error will be returned.

Parameters

[in]	clock_source	Clock source to initialize.
[in]	p_clock_cfg	Pointer to a structure that contains the dividers or multipliers to be used when configuring the PLL.

◆ clockStop

ssp_err_t(* cgc_api_t::clockStop) (cgc_clock_t clock_source)

Stop a clock.

Implemented as

- R_CGC_ClockStop()

Precondition

Clock to be stopped must not be stopped prior to calling this function or an error will be returned.

Parameters

[in]	clock_source	The clock source to stop.

◆ init

ssp_err_t(* cgc_api_t::init) (void)

Initial configuration

Implemented as

- R_CGC_Init()

Note

The BSP module calls this function at startup. No further initialization is necessary.

◆ lcdClockCfg

ssp_err_t(* cgc_api_t::lcdClockCfg) (cgc_clock_t clock)

Configure the segment LCD Clock (S3A7 and S124 only).

Implemented as

- R_CGC_LCDClockCfg()

Parameters

[in]	clock	Segment LCD clock source.

◆ lcdClockDisable

ssp_err_t(* cgc_api_t::lcdClockDisable) (void)

Disables the LCD clock (S3A7 and S124 only).

Implemented as

- R_CGC_LCDClockDisable()

◆ lcdClockEnable

ssp_err_t(* cgc_api_t::lcdClockEnable) (void)

Enable the LCD clock (S3A7 and S124 only).

Implemented as

- R_CGC_LCDClockEnable()

◆ oscStopDetect

ssp_err_t(* cgc_api_t::oscStopDetect) (void(*p_callback)(cgc_callback_args_t *p_args), bool enable)

Configure the Main Oscillator stop detection.

Implemented as

- R_CGC_OscStopDetect()

Parameters

[in]	p_callback	Callback function that will be called by the NMI interrupt when an oscillation stop is detected. If the second argument is "false", then this argument can be NULL.
[in]	enable	Enable/disable Oscillation Stop Detection.

◆ oscStopStatusClear

ssp_err_t(* cgc_api_t::oscStopStatusClear) (void)

Clear the oscillator stop detection flag.

Implemented as

- R_CGC_OscStopStatusClear()

◆ sdadcClockCfg

ssp_err_t(* cgc_api_t::sdadcClockCfg) (cgc_clock_t clock)

Configure the 24-bit Sigma-Delta A/D Converter Clock (S1JA only).

Implemented as

- R_CGC_SDADCClockCfg()

Parameters

[in]	clock	SDADC clock source.

◆ sdadcClockDisable

ssp_err_t(* cgc_api_t::sdadcClockDisable) (void)

Disables the SDADC clock (S1JA only).

Implemented as

- R_CGC_SDADCClockDisable()

◆ sdadcClockEnable

ssp_err_t(* cgc_api_t::sdadcClockEnable) (void)

Enable the SDADC clock (S1JA only).

Implemented as

- R_CGC_SDADCClockEnable()

◆ sdramClockOutDisable

ssp_err_t(* cgc_api_t::sdramClockOutDisable) (void)

Disables the SDRAM clock (S7G2 only).

Implemented as

- R_CGC_SDRAMClockOutDisable()

◆ sdramClockOutEnable

ssp_err_t(* cgc_api_t::sdramClockOutEnable) (void)

Enables the SDRAM clock output (S7G2 only).

Implemented as

- R_CGC_SDRAMClockOutEnable()

◆ systemClockFreqGet

ssp_err_t(* cgc_api_t::systemClockFreqGet) (cgc_system_clocks_t clock, uint32_t *p_freq_hz)

Return the frequency of the selected clock.

Implemented as

- R_CGC_SystemClockFreqGet()

Parameters

[in]	clock	Specifies the internal clock whose frequency is returned.
[out]	p_freq_hz	Returns the frequency in Hz referenced by this pointer.

◆ systemClockGet

ssp_err_t(* cgc_api_t::systemClockGet) (cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)

Get the system clock information.

Implemented as

- R_CGC_SystemClockGet()

Parameters

[in]	p_set_clock_cfg	Pointer to clock configuration structure
[out]	clock_source	Returns the current system clock.
[out]	p_clock_cfg	Returns the current system clock dividers.

◆ systemClockSet

ssp_err_t(* cgc_api_t::systemClockSet) (cgc_clock_t clock_source, cgc_system_clock_cfg_t const *const p_clock_cfg)

Set the system clock.

Implemented as

- R_CGC_SystemClockSet()

Precondition

The clock to be set as the system clock must be running prior to calling this function.

Parameters

[in]	clock_source	Clock source to set as system clock
[in]	p_clock_cfg	Pointer to the clock dividers configuration passed by the caller.

◆ systickUpdate

ssp_err_t(* cgc_api_t::systickUpdate) (uint32_t period_count, cgc_systick_period_units_t units)

Update the Systick timer.

Implemented as

- R_CGC_SystickUpdate()

Parameters

[in]	period_count	The duration for the systick period.
[in]	units	The units for the provided period.

◆ usbClockCfg

ssp_err_t(* cgc_api_t::usbClockCfg) (cgc_usb_clock_div_t divider)
Configures the USB clock (S7G2 only). **Implemented as** ◦ R_CGC_USBClockCfg() **Parameters**

[in]	divider	The divider of 3, 4 or 5, of the clock source.

◆ versionGet

ssp_err_t(* cgc_api_t::versionGet) (ssp_version_t *p_version)
Gets the CGC driver version. **Implemented as** ◦ R_CGC_VersionGet() **Parameters**

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_cgc_api.h

cgc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CGC Interface

#include <r_cgc_api.h>

Data Fields

cgc_clock_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
cgc_api_t const *	p_api

	Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_cgc_api.h

5.1.4.6 COMPARATOR Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for Comparators. More...

Data Structures

struct	comparator_info_t
struct	comparator_status_t
struct	comparator_callback_args_t
struct	comparator_cfg_t
struct	comparator_api_t
struct	comparator_instance_t

Macros

#define	COMPARATOR_API_VERSION_MAJOR (2U)

Typedefs

typedef void	comparator_ctrl_t

Enumerations

enum	comparator_mode_t { COMPARATOR_MODE_NORMAL = 0, COMPARATOR_MODE_WINDOW = 1 }
enum	comparator_trigger_t { COMPARATOR_TRIGGER_RISING = 1, COMPARATOR_TRIGGER_FALLING = 2, COMPARATOR_TRIGGER_BOTH_EDGE = 3 }

enum	comparator_polarity_invert_t { COMPARATOR_POLARITY_INVERT_OFF = 0, COMPARATOR_POLARITY_INVERT_ON = 1 }
enum	comparator_pin_output_t { COMPARATOR_PIN_OUTPUT_OFF = 0, COMPARATOR_PIN_OUTPUT_ON = 1 }
enum	comparator_filter_t { COMPARATOR_FILTER_OFF = 0, COMPARATOR_FILTER_1 = 4, COMPARATOR_FILTER_8 = 1, COMPARATOR_FILTER_16 = 2, COMPARATOR_FILTER_32 = 3 }
enum	comparator_state_t { COMPARATOR_STATE_OUTPUT_DISABLED = 0, COMPARATOR_STATE_OUTPUT_LOW = 1, COMPARATOR_STATE_OUTPUT_HIGH = 2 }

Detailed Description

Interface for Comparators.

Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

Implemented by: High-Speed Analog Comparator Low Power Analog Comparator

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

COMPARATOR Interface description: Comparator Driver on r_acmphs and HALACMPLPModule

Macro Definition Documentation

◆ COMPARATOR_API_VERSION_MAJOR

#define COMPARATOR_API_VERSION_MAJOR (2U)
Includes board and MCU related header files. Version Number of API.

Typedef Documentation

◆ comparator_ctrl_t

typedef void comparator_ctrl_t
Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. **Implemented as** ◦ acmphs_instance_ctrl_t ◦ acmplp_instance_ctrl_t

Enumeration Type Documentation

◆ comparator_filter_t

enum comparator_filter_t	
Comparator digital filtering sample clock divisor settings.	
Enumerator	
COMPARATOR_FILTER_OFF	Disable debounce filter.
COMPARATOR_FILTER_1	Filter using PCLK divided by 1, not supported by all implementations.
COMPARATOR_FILTER_8	Filter using PCLK divided by 8.
COMPARATOR_FILTER_16	Filter using PCLK divided by 16, not supported by all implementations.
COMPARATOR_FILTER_32	Filter using PCLK divided by 32.

◆ comparator_mode_t

enum comparator_mode_t	
Select whether to invert the polarity of the comparator output.	
Enumerator	
COMPARATOR_MODE_NORMAL	Normal mode.
COMPARATOR_MODE_WINDOW	Window mode, not supported by all implementations.

◆ comparator_pin_output_t

enum comparator_pin_output_t	
Select whether to include the comparator output on the output pin.	
Enumerator	
COMPARATOR_PIN_OUTPUT_OFF	Do not include comparator output on output pin.
COMPARATOR_PIN_OUTPUT_ON	Include comparator output on output pin.

◆ comparator_polarity_invert_t

enum comparator_polarity_invert_t	
Select whether to invert the polarity of the comparator output.	
Enumerator	
COMPARATOR_POLARITY_INVERT_OFF	Do not invert polarity.
COMPARATOR_POLARITY_INVERT_ON	Invert polarity.

◆ comparator_state_t

enum comparator_state_t	
Current comparator state.	
Enumerator	
COMPARATOR_STATE_OUTPUT_DISABLED	comparator_api_t::outputEnable() has not been called
COMPARATOR_STATE_OUTPUT_LOW	VCMP < VREF if polarity is not inverted, VCMP > VREF if inverted.
COMPARATOR_STATE_OUTPUT_HIGH	VCMP > VREF if polarity is not inverted, VCMP < VREF if inverted.

◆ comparator_trigger_t

enum comparator_trigger_t	
Trigger type: rising edge, falling edge, both edges, low level.	
Enumerator	
COMPARATOR_TRIGGER_RISING	Rising edge trigger.
COMPARATOR_TRIGGER_FALLING	Falling edge trigger.
COMPARATOR_TRIGGER_BOTH_EDGE	Both edges trigger.

comparator_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

uint32_t	min_stabilization_wait_us Minimum stabilization wait time in microseconds.

Detailed Description

Comparator information.

The documentation for this struct was generated from the following file:

- r_comparator_api.h

comparator_status_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

comparator_state_t	state Current comparator state.

Detailed Description

Comparator status.

The documentation for this struct was generated from the following file:

- r_comparator_api.h

comparator_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

void const *	p_context
uint32_t	channel The physical hardware channel that caused the interrupt.

Detailed Description

Callback function parameter data

Field Documentation

◆ p_context

void const* comparator_callback_args_t::p_context
Placeholder for user data. Set in comparator_api_t::open function in comparator_cfg_t.

The documentation for this struct was generated from the following file:

- r_comparator_api.h

comparator_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

uint8_t	channel Hardware channel used.
uint8_t	irq_ipl Interrupt priority.
comparator_mode_t	mode Normal or window mode.
comparator_trigger_t	trigger Trigger setting.
comparator_filter_t	filter Digital filter clock divisor setting.
comparator_polarity_invert_t	invert Whether to invert output.
comparator_pin_output_t	pin_output Whether to include output on output pin.
void(*	p_callback)(comparator_callback_args_t *p_args)
void const *	p_context
void const *	p_extend Comparator hardware dependent configuration.

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ p_callback

void(* comparator_cfg_t::p_callback) (comparator_callback_args_t *p_args)
Callback called when comparator event occurs.

◆ p_context

void const* comparator_cfg_t::p_context
Placeholder for user data. Passed to the user callback in comparator_callback_args_t.

The documentation for this struct was generated from the following file:

- r_comparator_api.h

comparator_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

ssp_err_t(*	open)(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)
ssp_err_t(*	outputEnable)(comparator_ctrl_t *const p_ctrl)
ssp_err_t(*	infoGet)(comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
ssp_err_t(*	statusGet)(comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
ssp_err_t(*	close)(comparator_ctrl_t *const p_ctrl)

ssp_err_t(* versionGet)(ssp_version_t *const p_version)

Detailed Description

Comparator functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)

Stop the comparator.

Implemented as

- R_ACMPHS_Close()
- R_ACMPLP_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ infoGet

ssp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

Implemented as

- R_ACMPHS_InfoGet()
- R_ACMPLP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_info	Comparator information stored here

◆ open

ssp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)

Initialize the comparator.

Implemented as

- R_ACMPHS_Open()
- R_ACMPLP_Open()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ outputEnable

ssp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)

Start the comparator.

Implemented as

- R_ACMPHS_OutputEnable()
- R_ACMPLP_OutputEnable()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ statusGet

ssp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)

Provide current comparator status.

Implemented as

- R_ACMPHS_StatusGet()
- R_ACMPLP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_status	Status stored here

◆ versionGet

ssp_err_t(* comparator_api_t::versionGet) (ssp_version_t *const p_version)

Retrieve the API version.

Implemented as

- R_ACMPHS_VersionGet()
- R_ACMPLP_VersionGet()

Precondition

This function retrieves the API version.

Parameters

[in]	p_version	Pointer to version structure

The documentation for this struct was generated from the following file:

- r_comparator_api.h

comparator_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » COMPARATOR Interface

#include <r_comparator_api.h>

Data Fields

comparator_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
comparator_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
comparator_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_comparator_api.h

5.1.4.7 CRC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for cyclic redundancy checking. More...

Data Structures

struct	crc_cfg_t
struct	crc_snoop_cfg_t
struct	crc_api_t
struct	crc_instance_t

Typedefs

typedef void	crc_ctrl_t

Enumerations

enum	crc_polynomial_t { CRC_POLYNOMIAL_CRC_8 = 1, CRC_POLYNOMIAL_CRC_16, CRC_POLYNOMIAL_CRC_CCITT, CRC_POLYNOMIAL_CRC_32, CRC_POLYNOMIAL_CRC_32C }
enum	crc_bit_order_t { CRC_BIT_ORDER_LMS_LSB = 0, CRC_BIT_ORDER_LMS_MSB }
enum	crc_snoop_direction_t { CRC_SNOOP_DIRECTION_RECEIVE = 0, CRC_SNOOP_DIRECTION_TRANSMIT }

Detailed Description

Interface for cyclic redundancy checking.

Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the

block using the CPU or by snooping on read or write activity on one of 10 SCI channels.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CRC Interface description: CRC Driver

Typedef Documentation

◆ crc_ctrl_t

typedef void crc_ctrl_t
CRC control block. Allocate an instance specific control block to pass into the CRC API calls. **Implemented as** ◦ crc_instance_ctrl_t

Enumeration Type Documentation

◆ crc_bit_order_t

enum crc_bit_order_t	
CRC Calculation Switching (LMS)	
Enumerator	
CRC_BIT_ORDER_LMS_LSB	Generates CRC for LSB first communication.
CRC_BIT_ORDER_LMS_MSB	Generates CRC for MSB first communication.

◆ crc_polynomial_t

enum crc_polynomial_t	
CRC Generating Polynomial Switching (GPS).	
Enumerator	
CRC_POLYNOMIAL_CRC_8	8-bit CRC-8 (X^8 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_16	16-bit CRC-16 (X^16 + X^15 + X^2 + 1)
CRC_POLYNOMIAL_CRC_CCITT	16-bit CRC-CCITT (X^16 + X^12 + X^5 + 1)
CRC_POLYNOMIAL_CRC_32	32-bit CRC-32 (X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_32C	32-bit CRC-32C (X^32 + X^28 + X^27 + X^26 + X^25 + X^23 + X^22 + X^20 + X^19 + X^18 + X^14 + X^13 + X^11 + X^10 + X^9 + X^8 + X^6 + 1)

◆ crc_snoop_direction_t

enum crc_snoop_direction_t	
Snoop-On-Write/Read Switch (CRCSWR)	
Enumerator	
CRC_SNOOP_DIRECTION_RECEIVE	Snoop-on-read.
CRC_SNOOP_DIRECTION_TRANSMIT	Snoop-on-write.

crc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CRC Interface

#include <r_crc_api.h>

Data Fields

crc_polynomial_t	polynomial CRC Generating Polynomial Switching. (GPS)

crc_bit_order_t	bit_order CRC Calculation Switching (LMS)
void const *	p_extend CRC Hardware Dependent Configuration.
bool	fifo_mode FIFO Mode selection for sci_uart in CRC snoop operation.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- r_crc_api.h

crc_snoop_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CRC Interface

#include <r_crc_api.h>

Data Fields

uint32_t	snoop_channel Register Snoop Address (CRCSA)
crc_snoop_direction_t	snoop_direction Snoop-On-Write/Read Switch (CRCSWR)

Detailed Description

Snoop configuration

The documentation for this struct was generated from the following file:

- r_crc_api.h

crc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CRC Interface

#include <r_crc_api.h>

Data Fields

ssp_err_t(*	open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
ssp_err_t(*	close)(crc_ctrl_t *const p_ctrl)
ssp_err_t(*	crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
ssp_err_t(*	snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
ssp_err_t(*	snoopDisable)(crc_ctrl_t *const p_ctrl)
ssp_err_t(*	snoopCfg)(crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)
ssp_err_t(*	calculate)(crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
ssp_err_t(*	versionGet)(ssp_version_t *version)

Detailed Description

CRC driver structure. General CRC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ calculate

ssp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)

Perform a CRC calculation on a block of data.

Implemented as

- R_CRC_Calculate()

Parameters

[in]	p_ctrl	Pointer to crc device handle.
[in]	input_buffer	A pointer to an array of data values.
[in]	num_bytes	The number of bytes (not elements) in the array.
[in]	crc_seed	The seeded value for crc calculations.
[out]	crc_result	The calculated value of the CRC calculation.

◆ close

ssp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)

Close the CRC module driver

Implemented as

- R_CRC_Close()

Parameters

[in]	p_ctrl	Pointer to crc device handle

Return values

SSP_SUCCESS	Configuration was successful.

◆ crcResultGet

ssp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

Return the current calculated value.

Implemented as

- R_CRC_CalculatedValueGet()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[out]	crc_result	The calculated value from the last CRC calculation.

◆ open

ssp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

Open the CRC driver module.

Implemented as

- R_CRC_Open()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	p_cfg	Pointer to a configuration structure.

◆ snoopCfg

ssp_err_t(* crc_api_t::snoopCfg) (crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)

Configure the snoop channel and direction.

Implemented as

- R_CRC_SnoopCfg()

Parameters

[in]	p_ctrl	Pointer to crc device handle.
[in]	p_snoopCfg	Snoop configuration.

◆ snoopDisable

ssp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)

Disable snooping.

Implemented as

- R_CRC_SnoopDisable()

Parameters

[in]	p_ctrl	Pointer to crc device handle.

◆ snoopEnable

ssp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

Enable snooping.

Implemented as

- R_CRC_SnoopEnable()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	crc_seed	CRC seed.

◆ versionGet

ssp_err_t(* crc_api_t::versionGet) (ssp_version_t *version)

Get the driver version based on compile time macros.

Implemented as

- R_CRC_VersionGet()

The documentation for this struct was generated from the following file:

- r_crc_api.h

crc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CRC Interface

#include <r_crc_api.h>

Data Fields

crc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
crc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
crc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_crc_api.h

5.1.4.8 Crypto Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Cryptographic algorithm APIs for encryption/decryption, signing/verification, and hashing. More...

Modules

	AES Interface AES encryption and decryption APIs.
	ARC4 Interface ARC4 encryption and decryption APIs.
	DSA Interface

	DSA (Digital Signature Algorithm) signature generation and verification APIs.
	ECC Interface ECC cryptographic functions for scalar multiplication, generate key, generate sign, verify sign and version get.
	HASH Algorithm Interface HASH_Interface Hash algorithm APIs.
	KEY_INSTALLATION Interface Key Installation functions for Key Installation procedure.
	RSA Interface RSA cryptographic functions for signature generation, verification, encryption and decryption.
	TDES Interface TDES encryption and decryption APIs.
	Random number generation RNG_Interface Random number generation.

Data Structures

struct	crypto_ctrl_t
struct	r_crypto_data_handle_t
struct	crypto_cfg_t
struct	crypto_interface_get_param_t
struct	crypto_api_t
struct	crypto_instance_t

Macros

#define	CRYPTO_API_VERSION_MAJOR (1U)

Enumerations

enum	crypto_word_endian_t
enum	crypto_algorithm_type_t
enum	crypto_hash_type_t
enum	crypto_key_type_t
enum	crypto_key_size_t
enum	crypto_chaining_mode_t
enum	crypto_module_status_t { CRYPTO_SCE_COMMON_MODULE_CLOSED = 0, CRYPTO_SCE_COMMON_MODULE_OPENED = 1 }

Detailed Description

Cryptographic algorithm APIs for encryption/decryption, signing/verification, and hashing.

Macro Definition Documentation

◆ CRYPTO_API_VERSION_MAJOR

#define CRYPTO_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Enumeration Type Documentation

◆ crypto_algorithm_type_t

enum crypto_algorithm_type_t
MIN and MAX values under enums are for internal use only Enumerated Crypto Algorithm Type

◆ crypto_chaining_mode_t

enum crypto_chaining_mode_t
Enumerated chaining modes

◆ crypto_hash_type_t

enum crypto_hash_type_t
Enumerated HASH Types

◆ crypto_key_size_t

enum crypto_key_size_t
Enumerated Key Sizes

◆ crypto_key_type_t

enum crypto_key_type_t
Enumerated Key Types

◆ crypto_module_status_t

enum crypto_module_status_t	
SCE Initialization status codes.	
Enumerator	
CRYPTO_SCE_COMMON_MODULE_CLOSED	The module is closed.
CRYPTO_SCE_COMMON_MODULE_OPENED	The module is opened.

◆ crypto_word_endian_t

enum crypto_word_endian_t
Enumerator for Crypto API uint32_t[] array word endian selection

AES Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

AES encryption and decryption APIs. More...

Data Structures

struct	aes_ctrl_t
struct	aes_cfg_t
struct	aes_api_t

struct	aes_instance_t

Macros

#define	AES_API_VERSION_MAJOR (1U)
#define	AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES (52)
#define	AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES (84)
#define	AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES (36)
#define	AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES (52)
#define	AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES (52)
#define	AES128_SECRET_KEY_SIZE_BYTES (16)
#define	AES192_SECRET_KEY_SIZE_BYTES (24)
#define	AES256_SECRET_KEY_SIZE_BYTES (32)

Variables

const aes_api_t	g_aes128ecb_on_sce
const aes_api_t	g_aes128cbc_on_sce
const aes_api_t	g_aes128ctr_on_sce
const aes_api_t	g_aes256ecb_on_sce
const aes_api_t	g_aes256cbc_on_sce
const aes_api_t	g_aes256ctr_on_sce
const aes_api_t	g_aes128gcm_on_sce
const aes_api_t	g_aes128xts_on_sce
const aes_api_t	g_aes256gcm_on_sce
const aes_api_t	g_aes256xts_on_sce
const aes_api_t	g_aes192ecb_on_sce
const aes_api_t	g_aes192cbc_on_sce
const aes_api_t	g_aes192ctr_on_sce

const aes_api_t	g_aes192gcm_on_sce
const aes_api_t	g_aes128ecb_on_sceHrk
const aes_api_t	g_aes192gcm_on_sceHrk

Detailed Description

AES encryption and decryption APIs.

Macro Definition Documentation

◆ AES128_SECRET_KEY_SIZE_BYTES

```
#define AES128_SECRET_KEY_SIZE_BYTES  (16)
```

Return AES secret key size in bytes for a 128-bit AES Key

◆ AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES

```
#define AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES  (36)
```

Return Wrapped AES secret key size in bytes for a 128-bit AES Key

◆ AES192_SECRET_KEY_SIZE_BYTES

```
#define AES192_SECRET_KEY_SIZE_BYTES  (24)
```

Return AES secret key size in bytes for a 192-bit AES Key

◆ AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES

```
#define AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES  (52)
```

Return Wrapped AES secret key size in bytes for a 192-bit AES Key

◆ AES256_SECRET_KEY_SIZE_BYTES

```
#define AES256_SECRET_KEY_SIZE_BYTES  (32)
```

Return AES secret key size in bytes for a 256-bit AES Key

◆ AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES

#define AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES (52)
Return Wrapped AES secret key size in bytes for a 256-bit AES Key

◆ AES_API_VERSION_MAJOR

#define AES_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

◆ AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES

#define AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES (52)
Return Wrapped AES-XTS secret key size in bytes for a 128-bit AES XTS Mode Key

◆ AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES

#define AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES (84)
Return AES-XTS secret key size in bytes for a 256-bit AES XTS Mode Key

Variable Documentation

◆ g_aes128cbc_on_sce

const aes_api_t g_aes128cbc_on_sce
AES 128-bit CBC mode implementation

◆ g_aes128ctr_on_sce

const aes_api_t g_aes128ctr_on_sce
AES 128-bit CTR mode implementation

◆ g_aes128ecb_on_sce

const aes_api_t g_aes128ecb_on_sce
AES interface available boards - S7G2, S5D9, S5D5, S3A7, S3A3 and S3A6 - Chaining modes CBC, GCM, CTR, ECB, XTS for 128 & 256-bit S7G2, S5D9, and S5D5 - Chaining modes CBC, GCM, CTR, ECB for 192-bit AES 128-bit ECB mode implementation

◆ g_aes128ecb_on_sceHrk

const aes_api_t g_aes128ecb_on_sceHrk
HRK Supported global structure definitions

◆ g_aes128gcm_on_sce

const aes_api_t g_aes128gcm_on_sce
AES 128-bit GCM mode implementation

◆ g_aes128xts_on_sce

const aes_api_t g_aes128xts_on_sce
AES 128-bit CCM mode implementation

◆ g_aes192cbc_on_sce

const aes_api_t g_aes192cbc_on_sce
AES 192-bit CBC mode implementation

◆ g_aes192ctr_on_sce

const aes_api_t g_aes192ctr_on_sce
AES 192-bit CTR mode implementation

◆ g_aes192ecb_on_sce

const aes_api_t g_aes192ecb_on_sce
AES 192-bit ECB mode implementation

◆ g_aes192gcm_on_sce

const aes_api_t g_aes192gcm_on_sce
AES 192-bit GCM mode implementation

◆ g_aes192gcm_on_sceHrk

const aes_api_t g_aes192gcm_on_sceHrk
AES 192-bit GCM HRK mode implementation

◆ g_aes256cbc_on_sce

const aes_api_t g_aes256cbc_on_sce
AES 256-bit CBC mode implementation

◆ g_aes256ctr_on_sce

const aes_api_t g_aes256ctr_on_sce
AES 256-bit CTR mode implementation

◆ g_aes256ecb_on_sce

const aes_api_t g_aes256ecb_on_sce
AES 256-bit ECB mode implementation

◆ g_aes256gcm_on_sce

const aes_api_t g_aes256gcm_on_sce
AES 256-bit GCM mode implementation

◆ g_aes256xts_on_sce

const aes_api_t g_aes256xts_on_sce
AES 256-bit XTS mode implementation

aes_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » AES Interface

#include <r_aes_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API
uint32_t	work_buffer [DRV_AES_CONTEXT_BUFFER_SIZE] Examples: AES-GCM mode uses this for storing authentication tag etc. More...

Detailed Description

AES Interface control structure

Field Documentation

◆ work_buffer

uint32_t aes_ctrl_t::work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]
Examples: AES-GCM mode uses this for storing authentication tag etc. used for storing context/state of the Cipher

The documentation for this struct was generated from the following file:

- r_aes_api.h

aes_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » AES Interface

#include <r_aes_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto engine api

Detailed Description

AES Interface configuration structure. User must fill in these values before invoking the open() function

---

The documentation for this struct was generated from the following file:

- r_aes_api.h

aes_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » AES Interface

#include <r_aes_api.h>

Data Fields

uint32_t(*	open)(aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)
uint32_t(*	createKey)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Generate an AES key for encrypt / decrypt operations. More...
uint32_t(*	encrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES encryption. More...
uint32_t(*	addAdditionalAuthenticationData)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source) Add additional authentication data (called before starting an encryption or decryption operation) More...
uint32_t(*	encryptFinal)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key,

	uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t output_num_words, uint32_t *p_dest) AES final encryption using the chaining mode and padding mode specified in the aes.open() function call.
uint32_t(*	decrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) AES Decryption. More...
uint32_t(*	setGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) set parameter specific to the mode More...
uint32_t(*	getGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get authentication tag data. More...
uint32_t(*	close)(aes_ctrl_t *const p_ctrl)
uint32_t(*	zeroPaddingEncrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) AES zero padding encryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -. More...
uint32_t(*	zeroPaddingDecrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) AES zero padding decryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -. More...
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

AES_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ addAdditionalAuthenticationData

uint32_t(* aes_api_t::addAdditionalAuthenticationData) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)

Add additional authentication data (called before starting an encryption or decryption operation)

Parameters

[in]	*p_key	pointer to the AES plain-text key
[in]	*p_iv	unused for ECB mode
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_source	input data buffer

◆ close

uint32_t(* aes_api_t::close) (aes_ctrl_t *const p_ctrl)

Close the AES module.

Parameters

[in]	p_ctrl	pointer to the control structure

◆ createKey

uint32_t(* aes_api_t::createKey) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key)

Generate an AES key for encrypt / decrypt operations.

Parameters

[in,out]	p_ctrl	pointer to control structure for the AES interface.
[in]	num_words	number of words in buffer p_key
[out]	p_key	pointer to key buffer. Generated key will be stored at this location.

◆ decrypt

uint32_t(* aes_api_t::decrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)

AES Decryption.

Decrypt input data with ECB mode using a 128-bit AES key

Parameters

[in]	*p_key	128-bit plain key
[in]	*p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	num_words	Size in words of p_source and p_dest data buffers. Each word is 4-bytes. Must be multiples of 4 words.
[in]	*p_source	input data buffer
[out]	*p_dest	output data buffer

◆ encrypt

uint32_t(* aes_api_t::encrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

AES encryption.

Encrypt input data with ECB mode using a 128-bit AES key

Parameters

[in]	*p_key	pointer to the AES plain-text key
[in]	*p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_source	input data buffer
[out]	*p_dest	output data buffer

◆ getGcmTag

uint32_t(* aes_api_t::getGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest)

Get authentication tag data.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	num_words	number of words in p_dest buffer. This must be atleast 4 words
[in]	p_dest	pointer to data buffer, must be of size 4 words.

◆ open

uint32_t(* aes_api_t::open) (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)

AES module open function. Must be called before performing any encrypt/decrypt operations.

Parameters

[in,out]	p_ctrl	pointer to control structure for the AES interface. Must be declared by user. Elements are set here.
[in]	p_cfg	pointer to control structure for the AES configuration. All elements of this structure must be set by user.

◆ setGcmTag

uint32_t(* aes_api_t::setGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)

set parameter specific to the mode

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	num_words	number of words in p_source buffer. This must be atleast 4 words
[in]	p_source	pointer to authentication tag data buffer, must be of size 4 words.

◆ versionGet

uint32_t(* aes_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

◆ zeroPaddingDecrypt

uint32_t(* aes_api_t::zeroPaddingDecrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)

AES zero padding decryption using the chaining mode and padding mode specified.
Implementation for GCM mode only API usage -.

1. Set expected tag value using the setGcmTag() function
2. Provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL
1. Decryption: set p_source to input encrypted data, decrypted data will be returned in p_dest
2. Verify the tag, invoke this API using p_source = NULL and p_dest = NULL, the return value indicates authentication tag verification status.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	*p_key	pointer to the AES plain-text key, the buffer size should be equal to the keylength
[in]	*p_iv	the buffer size must be 16 bytes
[in]	num_bytes	data buffer size in bytes.
[in]	*p_source	input data buffer, computes tag when set to NULL.
[out]	*p_dest	output data buffer, adds authentication data when set to NULL.

Note

this function is not thread safe.

◆ zeroPaddingEncrypt

uint32_t(* aes_api_t::zeroPaddingEncrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)

AES zero padding encryption using the chaining mode and padding mode specified.
Implementation for GCM mode only API usage -.

1. Provide any Add Authentication Data (AAD): set p_dest = NULL
2. Encryption: set p_source to input data and p_dest will return encrypted data
3. Get/Compute Tag: set p_source = NULL

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	*p_key	pointer to the AES plain-text key, the buffer size should be equal to the keylength
[in]	*p_iv	the buffer size must be 16 bytes
[in]	num_bytes	data buffer size in bytes.
[in]	*p_source	input data buffer, computes tag when set to NULL.
[out]	*p_dest	output data buffer, adds authentication data when set to NULL.

Note

this function is not thread safe.

The documentation for this struct was generated from the following file:

- r_aes_api.h

aes_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » AES Interface

#include <r_aes_api.h>

Data Fields

aes_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

aes_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
aes_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_aes_api.h

ARC4 Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

ARC4 encryption and decryption APIs. More...

Data Structures

struct	arc4_ctrl_t
struct	arc4_cfg_t
struct	arc4_api_t
struct	arc4_instance_t

Macros

#define	ARC4_API_VERSION_MAJOR (1U)

Variables

const arc4_api_t	g_arc4_on_sce

Detailed Description

ARC4 encryption and decryption APIs.

Macro Definition Documentation

◆ ARC4_API_VERSION_MAJOR

#define ARC4_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Variable Documentation

◆ g_arc4_on_sce

const arc4_api_t g_arc4_on_sce
ARC4 interface is only available on S7G2, S5D9 and S5D5. SCE/ARC4 implementation of ARC4 API.

arc4_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ARC4 Interface

#include <r_arc4_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API
uint32_t	state used to identify state of the ARC4 control block
bsp_lock_t	open indicates whether driver is opened with this control block More...

Detailed Description

ARC4 Interface control structure

Field Documentation

◆ open

bsp_lock_t arc4_ctrl_t::open
indicates whether driver is opened with this control block used for storing context of the cipher < ARC4 uses this for storing the sbox results for the next encrypt/ decrypt operations

---

The documentation for this struct was generated from the following file:

- r_arc4_api.h

arc4_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ARC4 Interface

#include <r_arc4_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto engine api
uint32_t	length Length of p_key in bytes.
uint8_t const *	p_key ARC4 key to use for encrypt or decrypto operations.

Detailed Description

ARC4 Interface configuration structure. User must fill in these values before invoking the open() function

---

The documentation for this struct was generated from the following file:

- r_arc4_api.h

arc4_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ARC4 Interface

#include <r_arc4_api.h>

Data Fields

uint32_t(*	open)(arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)
uint32_t(*	keySet)(arc4_ctrl_t *const p_ctrl, uint32_t length, uint8_t const *p_key)
uint32_t(*	arc4Process)(arc4_ctrl_t *const p_ctrl, uint32_t num_bytes, uint8_t *p_source, uint8_t *p_dest) Encrypt or decrypt source data p_source of length num_bytes and write the results to destination buffer p_dest More...
uint32_t(*	close)(arc4_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

ARC4_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ arc4Process

uint32_t(* arc4_api_t::arc4Process) (arc4_ctrl_t *const p_ctrl, uint32_t num_bytes, uint8_t *p_source, uint8_t *p_dest)

Encrypt or decrypt source data p_source of length num_bytes and write the results to destination buffer p_dest

Parameters

[in,out]	p_ctrl	pointer to control structure for ARC4 interface.
[in]	num_bytes	number of bytes to encrypt or decrypt, the value must be a multiple of 16
[in]	p_source	pointer to source data buffer
[out]	p_dest	pointer to destination data buffer

◆ close

uint32_t(* arc4_api_t::close) (arc4_ctrl_t *const p_ctrl)

Close the ARC4 module.

Parameters

[in,out]	p_ctrl	pointer to the control structure

◆ keySet

uint32_t(* arc4_api_t::keySet) (arc4_ctrl_t *const p_ctrl, uint32_t length, uint8_t const *p_key)

ARC4 module key set function. Resets the state of the ARC4 encryption block.

Parameters

[in,out]	p_ctrl	pointer to control structure for the ARC4 interface.
[in]	length	length of the p_key key material in bytes
[in]	p_key	pointer to the key material to use for encryption operations.

◆ open

uint32_t(* arc4_api_t::open) (arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)		
ARC4 module open function. Must be called before performing any encrypt/decrypt operations. Initializes the context for the encrypt or decrypt operations using the chosen Cipher interface.		
Parameters		
[in,out]	p_ctrl	pointer to control structure for the ARC4 interface. Must be declared by user. Elements are set here.
[in]	p_cfg	pointer to control structure for the ARC4 configuration. All elements of this structure must be set by user.

◆ versionGet

uint32_t(* arc4_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version.		
Parameters		
[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_arc4_api.h

arc4_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ARC4 Interface

#include <r_arc4_api.h>

Data Fields

arc4_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

arc4_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
arc4_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_arc4_api.h

DSA Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

DSA (Digital Signature Algorithm) signature generation and verification APIs. More...

Data Structures

struct	dsa_ctrl_t
struct	dsa_cfg_t
struct	dsa_api_t
struct	dsa_instance_t

Macros

#define	DSA_API_VERSION_MAJOR (1U)

Variables

const dsa_api_t	g_dsa1024_160_on_sce
const dsa_api_t	g_dsa2048_224_on_sce
const dsa_api_t	g_dsa2048_256_on_sce

Detailed Description

DSA (Digital Signature Algorithm) signature generation and verification APIs.

Macro Definition Documentation

◆ DSA_API_VERSION_MAJOR

#define DSA_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Variable Documentation

◆ g_dsa1024_160_on_sce

const dsa_api_t g_dsa1024_160_on_sce
DSA interface is only available on S7G2, S5D9 and S5D5. SCE/DSA implementation of DSA API.

◆ g_dsa2048_224_on_sce

const dsa_api_t g_dsa2048_224_on_sce
SCE/DSA implementation of DSA API.

◆ g_dsa2048_256_on_sce

const dsa_api_t g_dsa2048_256_on_sce
SCE/DSA implementation of DSA API.

dsa_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » DSA Interface

#include <r_dsa_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure

crypto_api_t const *	p_crypto_api pointer to crypto engine API

Detailed Description

DSA Interface control structure

The documentation for this struct was generated from the following file:

- r_dsa_api.h

dsa_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » DSA Interface

#include <r_dsa_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto engine api

Detailed Description

DSA Interface configuration structure. User must fill in these values before invoking the open() function

---

The documentation for this struct was generated from the following file:

- r_dsa_api.h

dsa_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » DSA Interface

#include <r_dsa_api.h>

Data Fields

uint32_t(*	open)(dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)
uint32_t(*	verify)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash) DSA signature verification using given DSA public key. More...
uint32_t(*	hashVerify)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash) DSA signature verification using given DSA public key. More...
uint32_t(*	sign)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest) DSA Signature generation using DSA private key. More...
uint32_t(*	hashSign)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest) DSA Signature generation using DSA private key. More...
uint32_t(*	close)(dsa_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

DSA_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* dsa_api_t::close) (dsa_ctrl_t *const p_ctrl)

Close the DSA module.

Parameters

[in]	p_ctrl	pointer to the control structure

◆ hashSign

uint32_t(* dsa_api_t::hashSign) (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)

DSA Signature generation using DSA private key.

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Parameters

[in]	*p_ctrl	pointer to control structure
[in]	*p_key	DSA plain text private key
[in]	*p_domain	pointer to DSA domain parameters.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_paddedHash	input data buffer
[out]	*p_dest	output data buffer

◆ hashVerify

uint32_t(* dsa_api_t::hashVerify) (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)

DSA signature verification using given DSA public key.

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Parameters

[in]	*p_ctrl	pointer to DSA control structure
[in]	*p_key	pointer to the DSA plain-text key.
[in]	*p_domain	pointer to DSA domain parameters.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_signature	signature data buffer to be verified
[in]	*p_paddedHash	padded hash of the input message

◆ open

uint32_t(* dsa_api_t::open) (dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)

DSA module open function. Must be called before performing any encrypt/decrypt operations.

Parameters

[in,out]	p_ctrl	pointer to control structure for the DSA interface. Must be declared by user. Elements are set here.
[in]	p_cfg	pointer to control structure for the DSA configuration. All elements of this structure must be set by user.

◆ sign

uint32_t(* dsa_api_t::sign) (const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)

DSA Signature generation using DSA private key.

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Parameters

[in]	*p_key	DSA plain text private key
[in]	*p_domain	pointer to DSA domain parameters.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_paddedHash	input data buffer
[out]	*p_dest	output data buffer

◆ verify

uint32_t(* dsa_api_t::verify) (const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)

DSA signature verification using given DSA public key.

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Parameters

[in]	*p_key	pointer to the DSA plain-text key.
[in]	*p_domain	pointer to DSA domain parameters.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_signature	signature data buffer to be verified
[in]	*p_paddedHash	padded hash of the input message

◆ versionGet

uint32_t(* dsa_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters** [out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_dsa_api.h

dsa_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » DSA Interface

#include <r_dsa_api.h>

Data Fields

dsa_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
dsa_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
dsa_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_dsa_api.h

ECC Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

ECC cryptographic functions for scalar multiplication, generate key, generate sign, verify sign and version get. More...

Data Structures

struct	ecc_ctrl_t
struct	ecc_cfg_t
struct	ecc_api_t
struct	ecc_instance_t

Macros

#define	ECC_API_VERSION_MAJOR (1U)

Variables

const ecc_api_t	g_ecc192_on_sce
const ecc_api_t	g_ecc192_on_sce_hrk
const ecc_api_t	g_ecc224_on_sce
const ecc_api_t	g_ecc224_on_sce_hrk
const ecc_api_t	g_ecc256_on_sce
const ecc_api_t	g_ecc256_on_sce_hrk
const ecc_api_t	g_ecc384_on_sce
const ecc_api_t	g_ecc384_on_sce_hrk

Detailed Description

ECC cryptographic functions for scalar multiplication, generate key, generate sign, verify sign and version get.

Macro Definition Documentation

◆ ECC_API_VERSION_MAJOR

#define ECC_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Variable Documentation

◆ g_ecc192_on_sce

const ecc_api_t g_ecc192_on_sce
ECC interface is only available on S7G2, S5D9 and S5D5. Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc192_on_sce_hrk

const ecc_api_t g_ecc192_on_sce_hrk
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc224_on_sce

const ecc_api_t g_ecc224_on_sce
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc224_on_sce_hrk

const ecc_api_t g_ecc224_on_sce_hrk
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc256_on_sce

const ecc_api_t g_ecc256_on_sce
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc256_on_sce_hrk

const ecc_api_t g_ecc256_on_sce_hrk
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc384_on_sce

const ecc_api_t g_ecc384_on_sce
Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc384_on_sce_hrk

const ecc_api_t g_ecc384_on_sce_hrk
Exported global variablesSCE/ECC implementation of ECC API.

ecc_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ECC Interface

#include <r_ecc_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl Pointer to crypto engine control structure.
crypto_api_t const *	p_crypto_api Pointer to crypto engine API.

Detailed Description

ECC Interface control structure.

The documentation for this struct was generated from the following file:

- r_ecc_api.h

ecc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ECC Interface

#include <r_ecc_api.h>

Data Fields

crypto_api_t const *	p_crypto_api Pointer to crypto engine API.

Detailed Description

ECC Interface configuration structure.

The documentation for this struct was generated from the following file:

- r_ecc_api.h

ecc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ECC Interface

#include <r_ecc_api.h>

Data Fields

ssp_err_t(*	open)(ecc_ctrl_t *const p_ctrl, ecc_cfg_t const *const p_cfg) Open the ECC driver. This API must be called before performing any ECC operations. More...
ssp_err_t(*	close)(ecc_ctrl_t *const p_ctrl) Close the ECC module. More...
ssp_err_t(*	scalarMultiplication)(ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) scalarMultiplication: This API calculates R=kP. More...
ssp_err_t(*	keyCreate)(ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public)

	keyCreate: This API generates key pair for ECC. More...
ssp_err_t(*	sign)(ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) sign: This API generates signature of ECDSA More...
ssp_err_t(*	verify)(ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) verify: This is a procedure for signature verification of ECDSA. More...
ssp_err_t(*	versionGet)(ssp_version_t *const p_version) versionGet: Gets version and stores it in provided pointer p_version. More...

Detailed Description

ECC_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* ecc_api_t::close) (ecc_ctrl_t *const p_ctrl)

Close the ECC module.

Parameters

[in]	p_ctrl	Pointer to the control structure.

◆ keyCreate

ssp_err_t(* ecc_api_t::keyCreate) (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public)

keyCreate: This API generates key pair for ECC.

Parameters

[in]	p_ctrl	Pointer to control structure for the ECC interface.
[in]	p_domain	a||b||p||n - These are domain parameters for ECC as defined in FIPS186-3.
[in]	p_generator_point	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
[in,out]	p_key_private	Private Key generated. Data length in words is updated in the output buffer data handle.
[in,out]	p_key_public	Public Key generated. Data length in words is updated in the output buffer data handle.

◆ open

ssp_err_t(* ecc_api_t::open) (ecc_ctrl_t *const p_ctrl, ecc_cfg_t const *const p_cfg)

Open the ECC driver. This API must be called before performing any ECC operations.

Parameters

[in]	p_ctrl	Pointer to control structure. Must be allocated by user before calling the API. Elements are set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user before calling this API.

◆ scalarMultiplication

ssp_err_t(* ecc_api_t::scalarMultiplication) (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r)

scalarMultiplication: This API calculates R=kP.

Parameters

[in]	p_ctrl	Pointer to control structure for the ECC interface.
[in]	p_domain	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
[in]	p_k	Scalar.
[in]	p_p	Px||Py, where, Px and Py are x and y coordinates respectively.
[in,out]	p_r	Rx||Ry (R=kP), where, Rx and Ry are x and y coordinates respectively. Data length in words is updated in the output buffer data handle.

◆ sign

ssp_err_t(* ecc_api_t::sign) (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s)

sign: This API generates signature of ECDSA

Parameters

[in]	p_ctrl	Pointer to control structure for the ECC interface.
[in]	p_domain	a||b||p||n - These are domain parameters for ECC as defined in IEEE1363.
[in]	p_generator_point	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
[in]	p_key_private	Private Key to process signature generation.
[in]	msg_digest	Message Digest. Length of this buffer must be equal to the ECC curve size in words.
[in,out]	signature_r	Signature r generated. Data length in words is updated in the output buffer data handle.
[in,out]	signature_s	Signature s generated. Data length in words is updated in the output buffer data handle.

◆ verify

ssp_err_t(* ecc_api_t::verify) (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s)

verify: This is a procedure for signature verification of ECDSA.

Parameters

[in]	p_ctrl	Pointer to control structure for the ECC interface.
[in]	p_domain	a||b||p||n - These are domain parameters for ECC as defined in IEEE1363.
[in]	p_generator_point	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
[in]	p_key_public	Public Key for signature verification.
[in]	msg_digest	Padded Message Digest. Length of this buffer must be equal to the ECC curve size in words.
[in]	signature_r	Signature r.
[in]	signature_s	Signature s.

◆ versionGet

ssp_err_t(* ecc_api_t::versionGet) (ssp_version_t *const p_version)

versionGet: Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_ecc_api.h

ecc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » ECC Interface

#include <r_ecc_api.h>

Data Fields

ecc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
ecc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
ecc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_ecc_api.h

HASH Algorithm Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

HASH_Interface Hash algorithm APIs. More...

Data Structures

struct	hash_cfg_t
struct	hash_ctrl_t
struct	hash_api_t

struct	hash_instance_t

Macros

#define	HASH_API_VERSION_MAJOR (1U)

Variables

const hash_api_t	g_md5_hash_on_sce
const hash_api_t	g_sha1_hash_on_sce
const hash_api_t	g_sha256_hash_on_sce

Detailed Description

HASH_Interface Hash algorithm APIs.

Macro Definition Documentation

◆ HASH_API_VERSION_MAJOR

#define HASH_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Variable Documentation

◆ g_md5_hash_on_sce

const hash_api_t g_md5_hash_on_sce
HASH interface is only available on S7G2, S5D9 and S5D5. MD5 implementation of HASH API.

◆ g_sha1_hash_on_sce

const hash_api_t g_sha1_hash_on_sce
SHA1 implementation of HASH API.

◆ g_sha256_hash_on_sce

const hash_api_t g_sha256_hash_on_sce
SHA256 implementation of HASH API.

hash_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » HASH Algorithm Interface

#include <r_hash_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API structure

Detailed Description

HASH_Interface configuration structure

The documentation for this struct was generated from the following file:

- r_hash_api.h

hash_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » HASH Algorithm Interface

#include <r_hash_api.h>

Data Fields

uint32_t	msgbuf [HASH_MESSAGE_BLOCK_SIZE_WORDS] message buffer to be hashed
uint32_t	hash [HASH_MAX_DIGEST_SIZE_WORDS] current hash value

uint64_t	length 64-bit message length (number of bits)
crypto_api_t const *	p_crypto_api pointer to crypto engine API

Detailed Description

HASH_Interface control structure

The documentation for this struct was generated from the following file:

- r_hash_api.h

hash_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » HASH Algorithm Interface

#include <r_hash_api.h>

Data Fields

uint32_t(*	open)(hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
uint32_t(*	updateHash)(const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
uint32_t(*	hashUpdate)(hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
uint32_t(*	close)(hash_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

HASH_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* hash_api_t::close) (hash_ctrl_t *const p_ctrl)

Close the hash module.

Parameters

[in]	p_ctrl	Pointer to the hash_ctrl_t control structure

◆ hashUpdate

uint32_t(* hash_api_t::hashUpdate) (hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)

update hash for the num_words words from source buffer p_source.

Parameters

[in]	p_ctrl	pointer to hash_ctrl_t control structure.
[in]	p_source	pointer to input message buffer. size must be a multiple of 64-bytes
[in]	num_words	number of words to be hashed from the buffer p_source
[in,out]	p_dest	pointer to the message digest. on input contains initialization value.

◆ open

uint32_t(* hash_api_t::open) (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)

HASH_Interface: Initial configuration

Precondition

HASH_Interface: Peripheral clocks and any required output pins should be configured prior to calling this function.

Parameters

[in,out]	HASH_Interface	p_ctrl Pointer to control structure. Must be declared by user. Elements set here.
[in]	HASH_Interface	p_cfg Pointer to configuration structure. All elements of this structure must be set by user.

◆ updateHash

uint32_t(* hash_api_t::updateHash) (const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)

update hash for the num_words words from source buffer p_source.

Parameters

[in]	p_source	pointer to input message buffer. size must be a multiple of 64-bytes
[in]	num_words	number of words to be hashed from the buffer p_source
[in,out]	p_dest	pointer to the message digest. on input contains initialization value.

◆ versionGet

uint32_t(* hash_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_hash_api.h

hash_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » HASH Algorithm Interface

#include <r_hash_api.h>

Data Fields

hash_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
hash_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
hash_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_hash_api.h

KEY_INSTALLATION Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

Key Installation functions for Key Installation procedure. More...

Data Structures

struct	key_installation_key_t
struct	key_installation_cfg_t
struct	key_installation_api_t
struct	key_installation_instance_t

Macros

#define	SCE_KEY_INSTALLATION_API_VERSION_MAJOR (2U)
#define	KEY_INSTALLATION_SESSION_KEY_SIZE_IN_WORDS (8U)
#define	KEY_INSTALLATION_RSA1024_MODULUS_SIZE_IN_WORDS (32U)
#define	KEY_INSTALLATION_RSA2048_MODULUS_SIZE_IN_WORDS (64U)
#define	KEY_INSTALLATION_RSA1024_ENCRYPTED_KEY_SIZE_IN_WORDS (36U)
#define	KEY_INSTALLATION_RSA2048_ENCRYPTED_KEY_SIZE_IN_WORDS (68U)
#define	KEY_INSTALLATION_RSA1024_WRAPPED_PRIVKEY_SIZE_IN_WORDS (37U)
#define	KEY_INSTALLATION_RSA2048_WRAPPED_PRIVKEY_SIZE_IN_WORDS (69U)
#define	KEY_INSTALLATION_AES128_ENCRYPTED_KEY_SIZE_IN_WORDS (8U)
#define	KEY_INSTALLATION_AES192_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_AES256_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_AES128_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_AES256_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS (20U)
#define	KEY_INSTALLATION_AES128_WRAPPED_KEY_SIZE_IN_WORDS (9U)
#define	KEY_INSTALLATION_AES192_WRAPPED_KEY_SIZE_IN_WORDS (13U)
#define	KEY_INSTALLATION_AES256_WRAPPED_KEY_SIZE_IN_WORDS (13U)

#define	KEY_INSTALLATION_AES128_WRAPPED_XTS_KEY_SIZE_IN_WORDS (13U)
#define	KEY_INSTALLATION_AES256_WRAPPED_XTS_KEY_SIZE_IN_WORDS (21U)
#define	KEY_INSTALLATION_ECC192_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_ECC224_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_ECC256_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
#define	KEY_INSTALLATION_ECC384_ENCRYPTED_KEY_SIZE_IN_WORDS (16U)
#define	KEY_INSTALLATION_ECC192_WRAPPED_KEY_SIZE_IN_WORDS (13U)
#define	KEY_INSTALLATION_ECC224_WRAPPED_KEY_SIZE_IN_WORDS (13U)
#define	KEY_INSTALLATION_ECC256_WRAPPED_KEY_SIZE_IN_WORDS (13U)
#define	KEY_INSTALLATION_ECC384_WRAPPED_KEY_SIZE_IN_WORDS (17U)

Typedefs

typedef void	key_installation_ctrl_t

Enumerations

enum	key_installation_key_format_t { KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_RSA_PRIVATE_KEY, KEY_INSTALLATION_KEY_FORMAT_WRAPPED_RSA_PRIVATE_KEY, KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_AES_KEY, KEY_INSTALLATION_KEY_FORMAT_WRAPPED_AES_KEY, KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_INSTALL_KEY, KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_ECC_PRIVATE_KEY, KEY_INSTALLATION_KEY_FORMAT_WRAPPED_ECC_PRIVATE_KEY, KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_RSA_PRIVATE_CRT_KEY, KEY_INSTALLATION_KEY_FORMAT_WRAPPED_RSA_PRIVATE_CRT_KEY, KEY_INSTALLATION_KEY_FORMAT_SESSION_KEY }
enum	key_installation_key_size_t { KEY_INSTALLATION_KEY_SIZE_RSA_1024, KEY_INSTALLATION_KEY_SIZE_RSA_2048, KEY_INSTALLATION_KEY_SIZE_AES_128, KEY_INSTALLATION_KEY_SIZE_AES_XTS_128,

```
  KEY_INSTALLATION_KEY_SIZE_AES_192,
KEY_INSTALLATION_KEY_SIZE_AES_256,
KEY_INSTALLATION_KEY_SIZE_AES_XTS_256,
KEY_INSTALLATION_KEY_SIZE_ENCRYPTED_INSTALL_416,
  KEY_INSTALLATION_KEY_SIZE_ECC_192,
KEY_INSTALLATION_KEY_SIZE_ECC_224,
KEY_INSTALLATION_KEY_SIZE_ECC_256,
KEY_INSTALLATION_KEY_SIZE_ECC_384,
  KEY_INSTALLATION_KEY_SIZE_SESSION
}
```

```
enum  key_installation_key_shared_index_t {
  KEY_INSTALLATION_KEY_SHARED_INDEX_0,
KEY_INSTALLATION_KEY_SHARED_INDEX_1,
KEY_INSTALLATION_KEY_SHARED_INDEX_2,
KEY_INSTALLATION_KEY_SHARED_INDEX_3,
  KEY_INSTALLATION_KEY_SHARED_INDEX_4,
KEY_INSTALLATION_KEY_SHARED_INDEX_5,
KEY_INSTALLATION_KEY_SHARED_INDEX_6,
KEY_INSTALLATION_KEY_SHARED_INDEX_7,
  KEY_INSTALLATION_KEY_SHARED_INDEX_8,
KEY_INSTALLATION_KEY_SHARED_INDEX_9,
KEY_INSTALLATION_KEY_SHARED_INDEX_A,
KEY_INSTALLATION_KEY_SHARED_INDEX_B,
  KEY_INSTALLATION_KEY_SHARED_INDEX_C,
KEY_INSTALLATION_KEY_SHARED_INDEX_D,
KEY_INSTALLATION_KEY_SHARED_INDEX_E,
KEY_INSTALLATION_KEY_SHARED_INDEX_F
}
```

Detailed Description

Key Installation functions for Key Installation procedure.

Macro Definition Documentation

◆ KEY_INSTALLATION_AES128_ENCRYPTED_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_AES128_ENCRYPTED_KEY_SIZE_IN_WORDS   (8U)
```

Macro definitions for AES Key sizes AES Encrypted key size in words for a 128-bit AES Key

◆ KEY_INSTALLATION_AES128_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_AES128_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS   (12U)
```

AES Encrypted key size in words for a 128-bit AES Key in XTS chaining mode

◆ KEY_INSTALLATION_AES128_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES128_WRAPPED_KEY_SIZE_IN_WORDS (9U)
AES Wrapped (output) key size in words for a 128-bit AES Key

◆ KEY_INSTALLATION_AES128_WRAPPED_XTS_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES128_WRAPPED_XTS_KEY_SIZE_IN_WORDS (13U)
AES Wrapped (output) key size in words for a 128-bit AES Key in XTS chaining mode

◆ KEY_INSTALLATION_AES192_ENCRYPTED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES192_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
AES Encrypted key size in words for a 192-bit AES Key

◆ KEY_INSTALLATION_AES192_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES192_WRAPPED_KEY_SIZE_IN_WORDS (13U)
AES Wrapped (output) key size in words for a 192-bit AES Key

◆ KEY_INSTALLATION_AES256_ENCRYPTED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES256_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
AES Encrypted key size in words for a 256-bit AES Key

◆ KEY_INSTALLATION_AES256_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES256_ENCRYPTED_XTS_KEY_SIZE_IN_WORDS (20U)
AES Encrypted key size in words for a 256-bit AES Key in XTS chaining mode

◆ KEY_INSTALLATION_AES256_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES256_WRAPPED_KEY_SIZE_IN_WORDS (13U)
AES Wrapped (output) key size in words for a 256-bit AES Key

◆ KEY_INSTALLATION_AES256_WRAPPED_XTS_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_AES256_WRAPPED_XTS_KEY_SIZE_IN_WORDS (21U)
AES Wrapped (output) key size in words for a 256-bit AES Key in XTS chaining mode

◆ KEY_INSTALLATION_ECC192_ENCRYPTED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC192_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
Macro definitions for ECC Key sizes ECC Encrypted key size in words for a 192-bit ECC Key

◆ KEY_INSTALLATION_ECC192_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC192_WRAPPED_KEY_SIZE_IN_WORDS (13U)
ECC Wrapped (output) key size in words for a 192-bit ECC Key

◆ KEY_INSTALLATION_ECC224_ENCRYPTED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC224_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
ECC Encrypted key size in words for a 224-bit ECC Key

◆ KEY_INSTALLATION_ECC224_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC224_WRAPPED_KEY_SIZE_IN_WORDS (13U)
ECC Wrapped (output) key size in words for a 224-bit ECC Key

◆ KEY_INSTALLATION_ECC256_ENCRYPTED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC256_ENCRYPTED_KEY_SIZE_IN_WORDS (12U)
ECC Encrypted key size in words for 256-bit ECC Key

◆ KEY_INSTALLATION_ECC256_WRAPPED_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_ECC256_WRAPPED_KEY_SIZE_IN_WORDS (13U)
ECC Wrapped (output) key size in words for a 256-bit ECC Key

◆ KEY_INSTALLATION_ECC384_ENCRYPTED_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_ECC384_ENCRYPTED_KEY_SIZE_IN_WORDS  (16U)
```

ECC Encrypted key size in words for a 384-bit ECC Key

◆ KEY_INSTALLATION_ECC384_WRAPPED_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_ECC384_WRAPPED_KEY_SIZE_IN_WORDS  (17U)
```

ECC Wrapped (output) key size in words for a 384-bit ECC Key

◆ KEY_INSTALLATION_RSA1024_ENCRYPTED_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_RSA1024_ENCRYPTED_KEY_SIZE_IN_WORDS  (36U)
```

RSA Encrypted key size in words for a 1024-bit RSA (private) Key

◆ KEY_INSTALLATION_RSA1024_MODULUS_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_RSA1024_MODULUS_SIZE_IN_WORDS  (32U)
```

Macro definitions for RSA Key sizes RSA Modulus size in words for a 1024-bit RSA Key

◆ KEY_INSTALLATION_RSA1024_WRAPPED_PRIVKEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_RSA1024_WRAPPED_PRIVKEY_SIZE_IN_WORDS  (37U)
```

RSA Wrapped (output) private key size in words for a 1024-bit RSA Key

◆ KEY_INSTALLATION_RSA2048_ENCRYPTED_KEY_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_RSA2048_ENCRYPTED_KEY_SIZE_IN_WORDS  (68U)
```

RSA Encrypted key size in words for a 2048-bit RSA (private) Key

◆ KEY_INSTALLATION_RSA2048_MODULUS_SIZE_IN_WORDS

```
#define KEY_INSTALLATION_RSA2048_MODULUS_SIZE_IN_WORDS  (64U)
```

RSA Modulus size in words for a 2048-bit RSA Key

◆ KEY_INSTALLATION_RSA2048_WRAPPED_PRIVKEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_RSA2048_WRAPPED_PRIVKEY_SIZE_IN_WORDS (69U)
RSA Wrapped (output) private key size in words for a 2048-bit RSA Key

◆ KEY_INSTALLATION_SESSION_KEY_SIZE_IN_WORDS

#define KEY_INSTALLATION_SESSION_KEY_SIZE_IN_WORDS (8U)
Macro definitions for universal KeyInstall Session/ IV Key sizes

◆ SCE_KEY_INSTALLATION_API_VERSION_MAJOR

#define SCE_KEY_INSTALLATION_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ key_installation_ctrl_t

typedef void key_installation_ctrl_t
Key Installation control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ key_installation_key_format_t

enum key_installation_key_format_t	
A structure to handle data for Key Installation module operation Supported key format definitions	
Enumerator	
KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_RSA_PRIVATE_KEY	Encrypted RSA Private key.
KEY_INSTALLATION_KEY_FORMAT_WRAPPED_RSA_PRIVATE_KEY	RSA Private Key wrapped.
KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_AES_KEY	Encrypted AES Private key.
KEY_INSTALLATION_KEY_FORMAT_WRAPPED_AES_KEY	AES Private Key wrapped.
KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_INSTALL_KEY	TO BE REMOVED w/ RSA/ ECC MODS.
KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_ECC_PRIVATE_KEY	Encrypted ECC Private key.
KEY_INSTALLATION_KEY_FORMAT_WRAPPED_ECC_PRIVATE_KEY	ECC Private Key wrapped.
KEY_INSTALLATION_KEY_FORMAT_ENCRYPTED_RSA_PRIVATE_CRT_KEY	Encrypted RSA Private CRT key.
KEY_INSTALLATION_KEY_FORMAT_WRAPPED_RSA_PRIVATE_CRT_KEY	RSA Private CRT Key wrapped.
KEY_INSTALLATION_KEY_FORMAT_SESSION_KEY	Session key for keyInstall API.

◆ key_installation_key_shared_index_t

enum key_installation_key_shared_index_t	
Supported Shared Key Index values (for keyInstall)	
Enumerator	
KEY_INSTALLATION_KEY_SHARED_INDEX_0	Shared Key Index 0.
KEY_INSTALLATION_KEY_SHARED_INDEX_1	Shared Key Index 1.
KEY_INSTALLATION_KEY_SHARED_INDEX_2	Shared Key Index 2.
KEY_INSTALLATION_KEY_SHARED_INDEX_3	Shared Key Index 3.
KEY_INSTALLATION_KEY_SHARED_INDEX_4	Shared Key Index 4.
KEY_INSTALLATION_KEY_SHARED_INDEX_5	Shared Key Index 5.
KEY_INSTALLATION_KEY_SHARED_INDEX_6	Shared Key Index 6.
KEY_INSTALLATION_KEY_SHARED_INDEX_7	Shared Key Index 7.
KEY_INSTALLATION_KEY_SHARED_INDEX_8	Shared Key Index 8.
KEY_INSTALLATION_KEY_SHARED_INDEX_9	Shared Key Index 9.
KEY_INSTALLATION_KEY_SHARED_INDEX_A	Shared Key Index 10.
KEY_INSTALLATION_KEY_SHARED_INDEX_B	Shared Key Index 11.
KEY_INSTALLATION_KEY_SHARED_INDEX_C	Shared Key Index 12.
KEY_INSTALLATION_KEY_SHARED_INDEX_D	Shared Key Index 13.
KEY_INSTALLATION_KEY_SHARED_INDEX_E	Shared Key Index 14.
KEY_INSTALLATION_KEY_SHARED_INDEX_F	Shared Key Index 15.

◆ key_installation_key_size_t

enum key_installation_key_size_t	
Supported key sizes	
Enumerator	
KEY_INSTALLATION_KEY_SIZE_RSA_1024	RSA 1024-bit key.
KEY_INSTALLATION_KEY_SIZE_RSA_2048	RSA 2048-bit key.
KEY_INSTALLATION_KEY_SIZE_AES_128	AES 128-bit key for CBC, CTR, ECB, GCM chaining modes.
KEY_INSTALLATION_KEY_SIZE_AES_XTS_128	AES 128-bit key for XTS chaining mode only.
KEY_INSTALLATION_KEY_SIZE_AES_192	AES 192-bit key for CBC, CTR, ECB, GCM chaining modes.
KEY_INSTALLATION_KEY_SIZE_AES_256	AES 256-bit key for CBC, CTR, ECB, GCM chaining modes.
KEY_INSTALLATION_KEY_SIZE_AES_XTS_256	AES 256-bit key for XTS chaining mode only.
KEY_INSTALLATION_KEY_SIZE_ENCRYPTED_INSTALL_416	Renesas provided install key size.
KEY_INSTALLATION_KEY_SIZE_ECC_192	ECC 192-bit key.
KEY_INSTALLATION_KEY_SIZE_ECC_224	ECC 224-bit key.
KEY_INSTALLATION_KEY_SIZE_ECC_256	ECC 256-bit key.
KEY_INSTALLATION_KEY_SIZE_ECC_384	ECC 384-bit key.
KEY_INSTALLATION_KEY_SIZE_SESSION	Session Key size for all operations.

key_installation_key_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » KEY_INSTALLATION Interface

#include <r_key_installation_api.h>

Data Fields

key_installation_key_format_t	key_format Indicates the key_format.
key_installation_key_size_t	key_size Indicates the key_type.
uint32_t *	p_data
uint32_t	data_length The length of data in WORDS(32-bits), pointed by p_data.

Detailed Description

Definition for Key data structure for Key Installation API operations

Field Documentation

◆ p_data

uint32_t* key_installation_key_t::p_data
Pointer to input (encrypted user key (OR) output data buffer to hold the wrapped key)

The documentation for this struct was generated from the following file:

- r_key_installation_api.h

key_installation_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » KEY_INSTALLATION Interface

#include <r_key_installation_api.h>

Data Fields

crypto_api_t const *	p_lower_lvl_crypto_api pointer to crypto engine api

void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

Key Installation Interface configuration structure. User must fill in these values before invoking the open() function

---

The documentation for this struct was generated from the following file:

- r_key_installation_api.h

key_installation_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » KEY_INSTALLATION Interface

#include <r_key_installation_api.h>

Data Fields

ssp_err_t(*	open)(key_installation_ctrl_t *const p_ctrl, key_installation_cfg_t const *const p_cfg)
ssp_err_t(*	keyInstall)(key_installation_ctrl_t *const p_ctrl, r_crypto_data_handle_t const *const p_user_key_rsa_modulus, key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data)
ssp_err_t(*	close)(key_installation_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

KEY_INSTALLATION_Interface functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* key_installation_api_t::close) (key_installation_ctrl_t *const p_ctrl)
Close API function of Key Installation module. **Parameters** [in] p_ctrl — Pointer to the control structure

◆ keyInstall

ssp_err_t(* key_installation_api_t::keyInstall) (key_installation_ctrl_t *const p_ctrl, r_crypto_data_handle_t const *const p_user_key_rsa_modulus, key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data)

Key installation API function that takes the user's encrypted key, a shared index, session key, and an IV then returns wrapped key of the user's plain-text key.

Parameters

[in]	p_ctrl	Pointer to the control structure.
[in]	p_user_key_rsa_modulus	Pointer to a user key's RSA modulus. Only applicable for Key Installation using RSA keys. For other crypto algorithms this parameter should be NULL.
[in]	p_user_key	Pointer to a user's encrypted key. In case of Key Installation using RSA keys, this will be the encrypted exponent.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to output wrapped output key. 'key_size' of p_key_data structure is UNUSED.

Note

For AES crypto algorithms, the buffer length required to hold the output wrapped key will be - 'Plaintext key word length + 5'. Expected values are defined by macros for each supported key length/ mode, for example KEY_INSTALLATION_AES128_WRAPPED_KEY_SIZE_IN_WORDS for 128-bit AES.

◆ open

ssp_err_t(* key_installation_api_t::open) (key_installation_ctrl_t *const p_ctrl, key_installation_cfg_t const *const p_cfg)
Key Installation module open API function. Must be called before invoking Key Installation operation. **Parameters** [in,out] p_ctrl — Pointer to control structure for the KEY_INSTALLATION interface. Must be declared by user. [in] p_cfg — Pointer to configuration structure for the KEY_INSTALLATION configuration. All elements of this structure must be set by user.

◆ versionGet

ssp_err_t(* key_installation_api_t::versionGet) (ssp_version_t *const p_version)
Gets version and stores it in provided pointer p_version. **Parameters** [out] p_version — Code and API version used.

The documentation for this struct was generated from the following file:

- r_key_installation_api.h

key_installation_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » KEY_INSTALLATION Interface

#include <r_key_installation_api.h>

Data Fields

key_installation_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
key_installation_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
key_installation_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_key_installation_api.h

RSA Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

RSA cryptographic functions for signature generation, verification, encryption and decryption. More...

Data Structures

struct	rsa_key_t
struct	rsa_ctrl_t
struct	rsa_cfg_t
struct	rsa_api_t
struct	rsa_instance_t

Macros

#define	RSA_API_VERSION_MAJOR (1U)

#define	RSA_MODULUS_SIZE_BYTES(RSA_SIZE_BITS) ((RSA_SIZE_BITS)/8U)
#define	RSA_PLAIN_TEXT_PUBLIC_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)32+(uint32_t)RSA_SIZE_BITS)/8U)
#define	RSA_PLAIN_TEXT_PRIVATE_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)2*(uint32_t)RSA_SIZE_BITS)/8U)
#define	RSA_PLAIN_TEXT_CRT_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)5*((uint32_t)RSA_SIZE_BITS))/16U)
#define	RSA_WRAPPPED_PRIVATE_KEY_SIZE_BYTES (RSA_SIZE_BITS) ((((uint32_t)2 * (uint32_t)RSA_SIZE_BITS)+(uint32_t)160)/8U)
#define	RSA_WRAPPPED_PRIVATE_CRT_KEY_SIZE_BYTES (RSA_SIZE_BITS) ((RSA_PLAIN_TEXT_CRT_KEY_SIZE_BYTES (RSA_SIZE_BITS))+20U)

Enumerations

enum	rsa_key_format_t { RSA_KEY_FORMAT_PLAIN_TEXT_PUBLIC_KEY, RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY, RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY, RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY, RSA_KEY_FORMAT_WRAPPED_PRIVATE_CRT_KEY }

Variables

const rsa_api_t	g_rsa1024_on_sce
const rsa_api_t	g_rsa1024_on_sce_hrk
const rsa_api_t	g_rsa2048_on_sce_hrk

Detailed Description

RSA cryptographic functions for signature generation, verification, encryption and decryption.

Macro Definition Documentation

◆ RSA_API_VERSION_MAJOR

#define RSA_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

◆ RSA_MODULUS_SIZE_BYTES

#define RSA_MODULUS_SIZE_BYTES (RSA_SIZE_BITS) ((RSA_SIZE_BITS)/8U)
Return RSA modulus size in bytes from the specified RSA modulus size in bits

◆ RSA_PLAIN_TEXT_CRT_KEY_SIZE_BYTES

#define RSA_PLAIN_TEXT_CRT_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)5*((uint32_t)RSA_SIZE_BITS))/16U)
Return RSA CRT private key size in bytes from the specified RSA modulus size in bits

◆ RSA_PLAIN_TEXT_PRIVATE_KEY_SIZE_BYTES

#define RSA_PLAIN_TEXT_PRIVATE_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)2*(uint32_t)RSA_SIZE_BITS)/8U)
Return RSA private key size in bytes from the specified RSA modulus size in bits

◆ RSA_PLAIN_TEXT_PUBLIC_KEY_SIZE_BYTES

#define RSA_PLAIN_TEXT_PUBLIC_KEY_SIZE_BYTES (RSA_SIZE_BITS) (((uint32_t)32+(uint32_t)RSA_SIZE_BITS)/8U)
Return RSA public key size in bytes from the specified RSA modulus size in bits

◆ RSA_WRAPPPED_PRIVATE_CRT_KEY_SIZE_BYTES

#define RSA_WRAPPPED_PRIVATE_CRT_KEY_SIZE_BYTES (RSA_SIZE_BITS) ((RSA_PLAIN_TEXT_CRT_KEY_SIZE_BYTES(RSA_SIZE_BITS))+20U)
Return RSA wrapped private CRT key size in bytes from the specified RSA modulus size in bits

◆ RSA_WRAPPPED_PRIVATE_KEY_SIZE_BYTES

#define RSA_WRAPPPED_PRIVATE_KEY_SIZE_BYTES (RSA_SIZE_BITS) ((((uint32_t)2 * (uint32_t)RSA_SIZE_BITS)+(uint32_t)160)/8U)
Return RSA wrapped private key size in bytes from the specified RSA modulus size in bits

Enumeration Type Documentation

◆ rsa_key_format_t

enum rsa_key_format_t	
RSA key format definitions	
Enumerator	
RSA_KEY_FORMAT_PLAIN_TEXT_PUBLIC_KEY	RSA public key in plain text format.
RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY	RSA private key in plain text format.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY	RSA CRT Key in plain text format.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY	RSA private Key wrapped using device specific key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_CRT_KEY	RSA private CRT Key wrapped using device specific key.

Variable Documentation

◆ g_rsa1024_on_sce

const rsa_api_t g_rsa1024_on_sce
RSA interface is only available on S7G2, S5D9 and S5D5. SCE/RSA implementation of RSA API.

◆ g_rsa1024_on_sce_hrk

const rsa_api_t g_rsa1024_on_sce_hrk
SCE/RSA implementation of RSA API.

◆ g_rsa2048_on_sce_hrk

const rsa_api_t g_rsa2048_on_sce_hrk
SCE/RSA implementation of RSA API.

rsa_key_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » RSA Interface

#include <r_rsa_api.h>

Data Fields

rsa_key_format_t	key_format Indicates if the key is in plain-text format or encrypted using device unique key.
uint32_t	length Length in bytes of the p_data buffer.
uint8_t *	p_data Buffer where the private key is stored.

Detailed Description

RSA key data structure

The documentation for this struct was generated from the following file:

- r_rsa_api.h

rsa_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » RSA Interface

#include <r_rsa_api.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API
uint32_t	stage_num processing stage

Detailed Description

RSA Interface control structure

The documentation for this struct was generated from the following file:

- r_rsa_api.h

rsa_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » RSA Interface

#include <r_rsa_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto engine api

Detailed Description

RSA Interface configuration structure. User must fill in these values before invoking the open() function

---

The documentation for this struct was generated from the following file:

- r_rsa_api.h

rsa_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » RSA Interface

#include <r_rsa_api.h>

Data Fields

uint32_t(*	open)(rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)
uint32_t(*	encrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const

	uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt source data from p_source using an RSA public key from p_key and write the results to destination buffer p_dest. More...
uint32_t(*	decrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. The RSA private key data p_key is specified in the standard format that consists of private exponent and the RSA modulus. The size of the private exponent and the RSA modulus is 1024-bits for the g_rsa1024_on_sce implementation and 2048-bits for the g_rsa2048_on_sce implementation. More...
uint32_t(*	decryptCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. RSA private key data is specified in CRT format. The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, starting with exponent2 at index 0. The size of each of these parameter is 512-bits for the g_rsa1024_on_sce implementation and 1024-bits for the g_rsa2048_on_sce implementation. More...
uint32_t(*	verify)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash) Verify signature given in buffer p_signature using the RSA public key p_key for the given padded message hash from buffer p_padded_hash. More...
uint32_t(*	sign)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest) Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. Write the results to the buffer p_dest. More...
uint32_t(*	signCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest) Generate signature for the given padded hash buffer p_padded_hash

	using the RSA private key p_key. RSA private key p_key is assumed to be in CRT format. Write the results to the buffer p_dest. The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, starting with exponent2 at index 0. The size of each of these parameter is 512-bits for the g_rsa1024_on_sce implementation and 1024-bits for the g_rsa2048_on_sce implementation. More...
uint32_t(*	close)(rsa_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)
uint32_t(*	keyCreate)(rsa_ctrl_t *const p_ctrl, rsa_key_t *p_private_key, rsa_key_t *p_public_key)

Detailed Description

RSA_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* rsa_api_t::close) (rsa_ctrl_t *const p_ctrl)

Close the RSA module.

Parameters

[in]	p_ctrl	pointer to the control structure

◆ decrypt

uint32_t(* rsa_api_t::decrypt) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. The RSA private key data p_key is specified in the standard format that consists of private exponent and the RSA modulus. The size of the private exponent and the RSA modulus is 1024-bits for the g_rsa1024_on_sce implementation and 2048-bits for the g_rsa2048_on_sce implementation.

Parameters

[in]	*p_ctrl	pointer to control structure for RSA interface
[in]	*p_key	pointer to RSA plain-text private key consisting of private exponent and the RSA modulus.
[in]	*p_domain	unused parameter for RSA decryption. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes.
[in]	*p_source	input data buffer to be decrypted.
[out]	*p_dest	output destination data buffer, decryption result will be stored here.

◆ decryptCrt

uint32_t(* rsa_api_t::decryptCrt) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. RSA private key data is specified in CRT format. The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, starting with exponent2 at index 0. The size of each of these parameter is 512-bits for the g_rsa1024_on_sce implementation and 1024-bits for the g_rsa2048_on_sce implementation.

Parameters

[in]	*p_key	pointer to RSA private key in CRT format.
[in]	*p_domain	unused parameter for RSA decryption. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes.
[in]	*p_source	input data buffer to be decrypted.
[out]	*p_dest	output destination data buffer, decryption result will be stored here.

◆ encrypt

uint32_t(* rsa_api_t::encrypt) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

Encrypt source data from p_source using an RSA public key from p_key and write the results to destination buffer p_dest.

Parameters

[in]	*p_ctrl	pointer to control structure for RSA interface
[in]	*p_key	pointer to the RSA plain-text public key consisting of 32-bit public exponent and RSA public modulus of size either 1024-bits or 2048-bits.
[in]	*p_domain	unused parameter for RSA encryption. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes.
[in]	*p_source	source data buffer to be encrypted.
[out]	*p_dest	destination data buffer, encryption result will be stored here.

◆ keyCreate

uint32_t(* rsa_api_t::keyCreate) (rsa_ctrl_t *const p_ctrl, rsa_key_t *p_private_key, rsa_key_t *p_public_key)

Generates an RSA key. This is a blocking call

Parameters

[in]	*p_ctrl	pointer to control structure for RSA interface
[in,out]	*p_private_key	pointer to a private key structure
[in,out]	*p_public_key	pointer to a public key structure

```
{
```

```
 // The following code snippet gives an example for generating and using a 1024-bit
RSA key.
 // For simplicity, the below code snippet does not check return values.
 rsa_ctrl_t rsa_ctrl;
 rsa_cfg_t rsa_cfg;
 rsa_key_t rsa_secret_key;
 rsa_key_t rsa_public_key;
    uint8_t    rsa_secret_key_data[RSA_PLAIN_TEXT_PRIVATE_KEY_SIZE_BYTES(1024)];
    uint8_t    rsa_public_key_data[RSA_PLAIN_TEXT_PUBLIC_KEY_SIZE_BYTES(1024)];
 // This example shows generation of an RSA private key in standard format.
 // To generate a CRT key, the key_format field should be set to
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY
 // and define rsa_secret_key_data buffer to be of size
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY
    rsa_secret_key.key_format = RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY;
    rsa_secret_key.length    = sizeof(rsa_secret_key_data);
    rsa_secret_key.p_data    = rsa_secret_key_data;
    rsa_public_key.key_format = RSA_KEY_FORMAT_PLAIN_TEXT_PUBLIC_KEY;
    rsa_public_key.length    = sizeof(rsa_public_key_data);
    rsa_publi_key.p_data      = rsa_public_key_data;
 g_rsa1024_on_sce.open(&rsa_ctrl, &rsa_cfg);
 g_rsa1024_on_sce.keyCreate(p_ctrl, &rsa_secret_key, &rsa_public_key);
 // p_source is a pointer to a padded hash data. The computed signature will be
stored at p_dest
 // The example below uses an RSA private key in standard format.
 // To compute signature using an RSA CRT private key, use the signCrt() interface
function.
 g_rsa1024_on_sce.sign(p_ctrl, num_words, rsa_secret_key.p_data, p_source, p_dest)
 g_rsa1024_on_sce.verify(p_ctrl, num_words, rsa_public_key.p_data, p_dest, p_source);
}
```

◆ open

uint32_t(* rsa_api_t::open) (rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)

RSA module open function. Must be called before performing any encrypt/decrypt or sign/verify operations.

Parameters

[in,out]	p_ctrl	pointer to control structure for the RSA interface. Must be declared by user. Elements are set here.
[in]	p_cfg	pointer to control structure for the RSA configuration. All elements of this structure must be set by user.

◆ sign

uint32_t(* rsa_api_t::sign) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)

Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. Write the results to the buffer p_dest.

Parameters

[in]	*p_ctrl	pointer to control structure for RSA interface
[in]	*p_key	pointer to RSA private key consisting of private exponent and the RSA modulus.
[in]	*p_domain	unused parameter. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_padded_hash	padded hash for the input message for which an RSA signature is desired
[out]	*p_dest	generated signature data will be written here.

◆ signCrt

uint32_t(* rsa_api_t::signCrt) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)

Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. RSA private key p_key is assumed to be in CRT format. Write the results to the buffer p_dest. The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, starting with exponent2 at index 0. The size of each of these parameter is 512-bits for the g_rsa1024_on_sce implementation and 1024-bits for the g_rsa2048_on_sce implementation.

Parameters

[in]	*p_ctrl	pointer to control structure for RSA interface
[in]	*p_key	pointer to RSA private key in CRT format.
[in]	*p_domain	unused parameter. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes. multiples of 4
[in]	*p_padded_hash	padded hash for the input message for which an RSA signature is desired
[out]	*p_dest	generated signature data will be written here.

◆ verify

uint32_t(* rsa_api_t::verify) (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash)

Verify signature given in buffer p_signature using the RSA public key p_key for the given padded message hash from buffer p_padded_hash.

Parameters

[in]	*p_key	pointer to the RSA plain-text public key consisting of 32-bit public exponent and RSA public modulus of size either 1024-bits or 2048-bits.
[in]	*p_domain	unused parameter. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each word is 4-bytes.
[in]	*p_signature	signature data that needs to be verified
[in]	*p_paddedHash	padded hash value of the input message buffer

◆ versionGet

uint32_t(* rsa_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_rsa_api.h

rsa_instance_t Struct Reference
Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » RSA Interface

#include <r_rsa_api.h>

Data Fields

rsa_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
rsa_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
rsa_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_rsa_api.h

TDES Interface

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

TDES encryption and decryption APIs. More...

Data Structures

struct	tdes_ctrl_t
struct	tdes_cfg_t
struct	tdes_api_t
struct	tdes_instance_t

Macros

#define	TDES_API_VERSION_MAJOR (1U)

Variables

const tdes_api_t	g_tdes192ecb_on_sce

Detailed Description

TDES encryption and decryption APIs.

Macro Definition Documentation

◆ TDES_API_VERSION_MAJOR

#define TDES_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

Variable Documentation

◆ g_tdes192ecb_on_sce

const tdes_api_t g_tdes192ecb_on_sce
TDES interface is only available on S7G2 and S5D9. SCE/TDES implementation of TDES API.

tdes_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » TDES Interface

#include <r_tdes_api.h>

Data Fields

crypto_ctrl_t	crypto_ctrl pointer to crypto control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API

Detailed Description

TDES Interface control structure

The documentation for this struct was generated from the following file:

- r_tdes_api.h

tdes_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » TDES Interface

#include <r_tdes_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto engine api

Detailed Description

TDES Interface configuration structure. User must fill in these values before invoking the open() function

The documentation for this struct was generated from the following file:

- r_tdes_api.h

tdes_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » TDES Interface

#include <r_tdes_api.h>

Data Fields

uint32_t(*	open)(tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg)
uint32_t(*	encrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) TDES encryption. More...
uint32_t(*	decrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

	TDES decryption. More...
uint32_t(*	close)(tdes_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

TDES_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* tdes_api_t::close) (tdes_ctrl_t *const p_ctrl)

Close the TDES module.

Parameters

[in]	p_ctrl	pointer to the control structure

◆ decrypt

uint32_t(* tdes_api_t::decrypt) (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

TDES decryption.

Decrypt input data with a 192-bit TDES key and the chaining mode specified.

Parameters

[in]	*p_key	pointer to the 192-bit plain-text key
[in,out]	*p_iv	pointer to initialization vector. Should be 8 bytes long. Unused for ECB mode. Next IV value is returned on successful completion.
[in]	num_words	Specifies the size of the input data in words. Should be multiples of 2. Note: 1 word is 4-bytes long.
[in]	*p_source	input data buffer - should be at least num_words long.
[out]	*p_dest	output data buffer - should be at least num_words long.

◆ encrypt

uint32_t(* tdes_api_t::encrypt) (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

TDES encryption.

Encrypt input data with a 192-bit TDES key and the chaining mode specified.

Parameters

[in]	*p_key	pointer to the TDES plain-text key.
[in,out]	*p_iv	pointer to initialization vector. Should be 8 bytes long. Unused for ECB mode. Next IV value is returned on successful completion.
[in]	num_words	Specifies the size of the input data in words. Should be multiples of 2. Note: 1 word is 4-bytes long.
[in]	*p_source	input data buffer - should be at least num_words long.
[out]	*p_dest	output data buffer - should be at least num_words long.

◆ open

uint32_t(* tdes_api_t::open) (tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg)

TDES module open function. Must be called before performing any encrypt/decrypt operations.

Parameters

[in,out]	p_ctrl	pointer to control structure for the TDES interface. Must be declared by user. Elements are set here.
[in]	p_cfg	pointer to control structure for the TDES configuration. All elements of this structure must be set by user.

◆ versionGet

uint32_t(* tdes_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters** [out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_tdes_api.h

tdes_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » TDES Interface

#include <r_tdes_api.h>

Data Fields

tdes_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
tdes_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
tdes_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_tdes_api.h

Random number generation

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

RNG_Interface Random number generation. More...

Data Structures

struct	trng_ctrl_t
struct	trng_cfg_t
struct	trng_api_t
struct	trng_instance_t

Macros

#define	TRNG_API_VERSION_MAJOR (1U)

Detailed Description

RNG_Interface Random number generation.

Macro Definition Documentation

◆ TRNG_API_VERSION_MAJOR

#define TRNG_API_VERSION_MAJOR (1U)
Register definitions, common services and error codes.

trng_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » Random number generation

#include <r_trng_api.h>

Data Fields

uint32_t	nattempts number of retries

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto control structure
crypto_api_t const *	p_crypto_api pointer to crypto-engine API
uint32_t	prevbuf [TRNG_REGISTER_SIZE_WORDS] previous random data
uint32_t	currbuf [TRNG_REGISTER_SIZE_WORDS] current random data

Detailed Description

TRNG_Interface control structure.

The documentation for this struct was generated from the following file:

- r_trng_api.h

trng_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » Random number generation

#include <r_trng_api.h>

Data Fields

crypto_api_t const *	p_crypto_api pointer to crypto API
uint32_t	nattempts number of retries when a continuous test failure occurs

Detailed Description

TRNG interface configuration parameters

The documentation for this struct was generated from the following file:

- r_trng_api.h

trng_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » Random number generation

#include <r_trng_api.h>

Data Fields

uint32_t(*	open)(trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
uint32_t(*	read)(trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)
uint32_t(*	close)(trng_ctrl_t *const p_ctrl)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

TRNG_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* trng_api_t::close) (trng_ctrl_t *const p_ctrl)

Close the TRNG interface driver

Parameters

[in]	p_ctrl	pointer to trng interface control structure

◆ open

uint32_t(* trng_api_t::open) (trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)

Open the TRNG driver for reading random data from the hardware TRNG module

Parameters

[in,out]	p_ctrl	Pointer to control structure. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ read

uint32_t(* trng_api_t::read) (trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)

Generate nwords of random number words and store them in p_rngbuf buffer

Parameters

[in]	p_ctrl	pointer to trng control structure
[out]	p_rngbuf	generated random numbers will be stored to the buffer p_rngbuf
[in]	nwords	number of random words to generate

◆ versionGet

uint32_t(* trng_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_trng_api.h

trng_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface » Random number generation

#include <r_trng_api.h>

Data Fields

trng_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
trng_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
trng_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_trng_api.h

crypto_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Data Fields

uint32_t	state indicates state of the SCE/SCE-Lite driver e.g whether it is initialized

Detailed Description

Crypto_Interface Add API definitions required by user here.

The documentation for this struct was generated from the following file:

- r_crypto_api.h

r_crypto_data_handle_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Data Fields

uint32_t *	p_data Pointer to data.
uint32_t	data_length The length of data pointed by p_data.

Detailed Description

A structure to handle data among Crypto HAL modules

The documentation for this struct was generated from the following file:

- r_crypto_api.h

crypto_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Data Fields

void(*	p_sce_long_plg_end_callback)(void)

crypto_word_endian_t	endian_flag
void *	p_sce_api_interfaces

Detailed Description

Crypto engine configuration parameters

Field Documentation

◆ endian_flag

crypto_word_endian_t crypto_cfg_t::endian_flag
Endian flag, indicates word endianness for the uint32_t[] array inputs used in Crypto APIs

◆ p_sce_api_interfaces

void* crypto_cfg_t::p_sce_api_interfaces
Pointer to the structure containing crypto API interfaces available to the selected MCU.

◆ p_sce_long_plg_end_callback

void(* crypto_cfg_t::p_sce_long_plg_end_callback) (void)
Callback provided when a ISR occurs. Set to NULL for no CPU interrupt.

The documentation for this struct was generated from the following file:

- r_crypto_api.h

crypto_interface_get_param_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Detailed Description

Parameters for requesting HAL API interface object

The documentation for this struct was generated from the following file:

- r_crypto_api.h

crypto_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Data Fields

uint32_t(*	open)(crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg)
uint32_t(*	close)(crypto_ctrl_t *const p_ctrl)
uint32_t(*	interfaceGet)(crypto_interface_get_param_t *const interface_info, void *const p_interface)
uint32_t(*	statusGet)(uint32_t *p_status)
uint32_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Crypto_Interface SCE functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

uint32_t(* crypto_api_t::close) (crypto_ctrl_t *const p_ctrl)

Close the crypto interface module for the given control structure p_ctrl

Parameters

[in]	p_ctrl	pointer to control structure

◆ interfaceGet

uint32_t(* crypto_api_t::interfaceGet) (crypto_interface_get_param_t *const interface_info, void *const p_interface)

Get API interface structure object based on the interface_info provided.

Parameters

[in]	interface_info	pointer to structure containing requested interface information.
[out]	p_interface	pointer whose value points to interface structure object.

Note

p_interface must be of pointer type and its address must be passed in this API. Passing the pointer and not its address may result in undefined behavior at run-time.
Value of p_interface is allowed to be passed as NULL at the time of API call.

◆ open

uint32_t(* crypto_api_t::open) (crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg)

Open crypto module using the given configuration

Parameters

[in,out]	p_ctrl	pointer to control structure. Must be declared by user. Elements set here.
[in]	p_cfg	pointer to configuration structure. All elements of this structure must be set by user

◆ statusGet

uint32_t(* crypto_api_t::statusGet) (uint32_t *p_status)

Get status of SCE initialization

Parameters

[out]	p_status	initialization status of SCE module will be written to the memory pointed to by p_status

◆ versionGet

uint32_t(* crypto_api_t::versionGet) (ssp_version_t *const p_version)		
Gets version and stores it in provided pointer p_version. **Parameters**		
[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_crypto_api.h

crypto_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Crypto Interface

#include <r_crypto_api.h>

Data Fields

crypto_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
crypto_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
crypto_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_crypto_api.h

5.1.4.9 CTSU v2 Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for Capacitive Touch Controllers. More...

Data Structures

struct	ctsu_callback_args_t
struct	ctsu_element_cfg_t
struct	ctsu_cfg_t
struct	ctsu_api_t
struct	ctsu_instance_t

Typedefs

typedef void	ctsu_ctrl_t

Enumerations

enum	ctsu_event_t { CTSU_EVENT_SCAN_COMPLETE = 0x00U, CTSU_EVENT_OVERFLOW = 0x01U, CTSU_EVENT_ICOMP = 0x02U }
enum	ctsu_cap_t { CTSU_CAP_SOFTWARE, CTSU_CAP_EXTERNAL }
enum	ctsu_atune1_t { CTSU_ATUNE1_NORMAL = 0U, CTSU_ATUNE1_HIGH }
enum	ctsu_md_t { CTSU_MODE_SELF_MULTI_SCAN = 1U, CTSU_MODE_MUTUAL_FULL_SCAN = 3U, CTSU_MODE_DIAGNOSIS_SCAN = 33 }
enum	ctsu_ssdiv_t { CTSU_SSDIV_4000, CTSU_SSDIV_2000, CTSU_SSDIV_1330, CTSU_SSDIV_1000, CTSU_SSDIV_0800, CTSU_SSDIV_0670, CTSU_SSDIV_0570, CTSU_SSDIV_0500, CTSU_SSDIV_0440, CTSU_SSDIV_0400, CTSU_SSDIV_0360, CTSU_SSDIV_0330, CTSU_SSDIV_0310, CTSU_SSDIV_0290, CTSU_SSDIV_0270, CTSU_SSDIV_0000 }

Detailed Description

Interface for Capacitive Touch Controllers.

Summary

The CTSU v2 interface provides the functionality necessary to open, close, run and control the CTSU depending upon the configuration passed as arguments.

Implemented by: CTSU v2

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CTSU v2 Interface description: CTSU v2 Driver

Typedef Documentation

◆ ctsu_ctrl_t

typedef void ctsu_ctrl_t
CTSU Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ ctsu_instance_ctrl_t

Enumeration Type Documentation

◆ ctsu_atune1_t

enum ctsu_atune1_t	
CTSU Power Supply Capacity Adjustment	
Enumerator	
CTSU_ATUNE1_NORMAL	Normal output (40uA)
CTSU_ATUNE1_HIGH	High-current output (80uA)

◆ ctsu_cap_t

enum ctsu_cap_t	
CTSU Scan Start Trigger Select	
Enumerator	
CTSU_CAP_SOFTWARE	Scan start by software trigger.
CTSU_CAP_EXTERNAL	Scan start by external trigger.

◆ ctsu_event_t

enum ctsu_event_t	
CTSU Events for callback function	
Enumerator	
CTSU_EVENT_SCAN_COMPLETE	Normal end.
CTSU_EVENT_OVERFLOW	Sensor counter overflow (CTSUST.CTSUSOVF set)
CTSU_EVENT_ICOMP	Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set)

◆ ctsu_md_t

enum ctsu_md_t	
CTSU Measurement Mode Select	
Enumerator	
CTSU_MODE_SELF_MULTI_SCAN	Self-capacitance multi scan mode.
CTSU_MODE_MUTUAL_FULL_SCAN	Mutual capacitance full scan mode.
CTSU_MODE_DIAGNOSIS_SCAN	Diagnosis scan mode.

◆ ctsu_ssdiv_t

enum ctsu_ssdiv_t	
CTSU Spectrum Diffusion Frequency Division Setting	
Enumerator	
CTSU_SSDIV_4000	4.00 <= Base clock frequency (MHz)
CTSU_SSDIV_2000	2.00 <= Base clock frequency (MHz) < 4.00
CTSU_SSDIV_1330	1.33 <= Base clock frequency (MHz) < 2.00
CTSU_SSDIV_1000	1.00 <= Base clock frequency (MHz) < 1.33
CTSU_SSDIV_0800	0.80 <= Base clock frequency (MHz) < 1.00
CTSU_SSDIV_0670	0.67 <= Base clock frequency (MHz) < 0.80
CTSU_SSDIV_0570	0.57 <= Base clock frequency (MHz) < 0.67
CTSU_SSDIV_0500	0.50 <= Base clock frequency (MHz) < 0.57
CTSU_SSDIV_0440	0.44 <= Base clock frequency (MHz) < 0.50
CTSU_SSDIV_0400	0.40 <= Base clock frequency (MHz) < 0.44
CTSU_SSDIV_0360	0.36 <= Base clock frequency (MHz) < 0.40
CTSU_SSDIV_0330	0.33 <= Base clock frequency (MHz) < 0.36
CTSU_SSDIV_0310	0.31 <= Base clock frequency (MHz) < 0.33
CTSU_SSDIV_0290	0.29 <= Base clock frequency (MHz) < 0.31
CTSU_SSDIV_0270	0.27 <= Base clock frequency (MHz) < 0.29
CTSU_SSDIV_0000	0.00 <= Base clock frequency (MHz) < 0.27

ctsu_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CTSU v2 Interface

#include <r_ctsuv2_api.h>

Data Fields

ctsu_event_t	event The event can be used to identify what caused the callback.
void const *	p_context Placeholder for user data. Set in ctsu_api_t::open function in ctsu_cfg_t.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_ctsuv2_api.h

ctsu_element_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CTSU v2 Interface

#include <r_ctsuv2_api.h>

Data Fields

ctsu_ssdiv_t	ssdiv CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)
uint16_t	so CTSU Sensor Offset Adjustment.
uint8_t	snum CTSU Measurement Count Setting.
uint8_t	sdpa CTSU Base Clock Setting.

Detailed Description

CTSU Configuration parameters. Element Configuration

The documentation for this struct was generated from the following file:

- r_ctsuv2_api.h

ctsu_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CTSU v2 Interface

#include <r_ctsuv2_api.h>

Data Fields

ctsu_cap_t	cap CTSU Scan Start Trigger Select.
ctsu_atune1_t	atune1 CTSU Power Supply Capacity Adjustment.
ctsu_md_t	md CTSU Measurement Mode Select.
uint8_t	ctsuchac0 TS00-TS07 enable mask.
uint8_t	ctsuchac1 TS08-TS15 enable mask.
uint8_t	ctsuchac2 TS16-TS23 enable mask.
uint8_t	ctsuchac3 TS24-TS31 enable mask.

uint8_t	ctsuchac4 TS32-TS39 enable mask.
uint8_t	ctsuchtrc0 TS00-TS07 mutual-tx mask.
uint8_t	ctsuchtrc1 TS08-TS15 mutual-tx mask.
uint8_t	ctsuchtrc2 TS16-TS23 mutual-tx mask.
uint8_t	ctsuchtrc3 TS24-TS31 mutual-tx mask.
uint8_t	ctsuchtrc4 TS32-TS39 mutual-tx mask.
ctsu_element_cfg_t const *	p_elements Pointer to elements configuration array.
uint8_t	num_rx Number of receive terminals.
uint8_t	num_tx Number of transmit terminals.
uint16_t	num_moving_average Number of moving average for measurement data.
bool	tunning_enable Initial offset tuning flag.
void(*	p_callback)(ctsu_callback_args_t *p_args)

	Callback provided when CTSUFN ISR occurs.
transfer_instance_t const *	p_transfer_tx DTC instance for transmit at CTSUWR. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx DTC instance for receive at CTSURD. Set to NULL if unused.
IRQn_Type	write_irq CTSU_CTSUWR interrupt vector.
IRQn_Type	read_irq CTSU_CTSURD interrupt vector.
IRQn_Type	end_irq CTSU_CTSUFN interrupt vector.
void const *	p_context User defined context passed into callback function.
void const *	p_extend Pointer to extended configuration by instance of interface.
uint16_t	tuning_self_target_value Target self value for initial offset tuning.
uint16_t	tuning_mutual_target_value Target mutual value for initial offset tuning.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- r_ctsuv2_api.h

ctsu_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CTSU v2 Interface

#include <r_ctsuv2_api.h>

Data Fields

ssp_err_t(*	open)(ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)
ssp_err_t(*	scanStart)(ctsu_ctrl_t *const p_ctrl)
ssp_err_t(*	dataGet)(ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)
ssp_err_t(*	diagnosis)(ctsu_ctrl_t *const p_ctrl)
ssp_err_t(*	callbackSet)(ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)
ssp_err_t(*	close)(ctsu_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_data)

Detailed Description

Functions implemented at the HAL layer will follow this API.

Field Documentation

◆ callbackSet

ssp_err_t(* ctsu_api_t::callbackSet) (ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_CTSU_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the CTSU control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

ssp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- R_CTSU_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ dataGet

ssp_err_t(* ctsu_api_t::dataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)

Data get.

Implemented as

- R_CTSU_DataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Pointer to get data array.

◆ diagnosis

ssp_err_t(* ctsu_api_t::diagnosis) (ctsu_ctrl_t *const p_ctrl)

Diagnosis.

Implemented as

- R_CTSU_Diagnosis()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ open

ssp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)

Open driver.

Implemented as

- R_CTSU_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ scanStart

ssp_err_t(* ctsu_api_t::scanStart) (ctsu_ctrl_t *const p_ctrl)

Scan start.

Implemented as

- R_CTSU_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ versionGet

ssp_err_t(* ctsu_api_t::versionGet) (ssp_version_t *const p_data)

Return the version of the driver.

Implemented as

- R_CTSU_VersionGet()

Parameters

[out]	p_data	Memory address to return version information to.

The documentation for this struct was generated from the following file:

- r_ctsuv2_api.h

ctsu_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » CTSU v2 Interface

#include <r_ctsuv2_api.h>

Data Fields

ctsu_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
ctsu_cfg_t const *	p_cfg

	Pointer to the configuration structure for this instance.
ctsu_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_ctsuv2_api.h

5.1.4.10 DAC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for D/A converters. More...

Data Structures

struct	dac_info_t
struct	dac_cfg_t
struct	dac_api_t
struct	dac_instance_t

Typedefs

typedef uint16_t	dac_size_t
typedef void	dac_ctrl_t

Enumerations

enum	dac_data_format_t { DAC_DATA_FORMAT_FLUSH_RIGHT = 0, DAC_DATA_FORMAT_FLUSH_LEFT = 1 }

Detailed Description

Interface for D/A converters.

Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

DAC Interface description: DAC Driver

Typedef Documentation

◆ dac_ctrl_t

typedef void dac_ctrl_t
DAC control block. Allocate an instance specific control block to pass into the DAC API calls. **Implemented as** ◦ dac_instance_ctrl_t

◆ dac_size_t

typedef uint16_t dac_size_t
Data type to store DAC output value.

Enumeration Type Documentation

◆ dac_data_format_t

enum dac_data_format_t	
DAC Open API AD/DA data format settings.	
Enumerator	
DAC_DATA_FORMAT_FLUSH_RIGHT	LSB of data is flush to the right leaving the top 4 bits unused.
DAC_DATA_FORMAT_FLUSH_LEFT	MSB of data is flush to the left leaving the bottom 4 bits unused.

dac_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DAC Interface

#include <r_dac_api.h>

Data Fields

uint8_t	bit_width Resolution of the DAC.

Detailed Description

DAC information structure to store various information for a DAC

The documentation for this struct was generated from the following file:

- r_dac_api.h

dac_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DAC Interface

#include <r_dac_api.h>

Data Fields

uint8_t	channel ID associated with this DAC channel.
bool	ad_da_synchronized AD/DA synchronization.
dac_data_format_t	data_format Data format.
bool	output_amplifier_enabled Output amplifier enable.

Detailed Description

DAC Open API configuration parameter

The documentation for this struct was generated from the following file:

- r_dac_api.h

dac_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DAC Interface

#include <r_dac_api.h>

Data Fields

ssp_err_t(*	open)(dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
ssp_err_t(*	close)(dac_ctrl_t *p_ctrl)
ssp_err_t(*	write)(dac_ctrl_t *p_ctrl, dac_size_t value)
ssp_err_t(*	start)(dac_ctrl_t *p_ctrl)
ssp_err_t(*	stop)(dac_ctrl_t *p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)
ssp_err_t(*	infoGet)(dac_info_t *const p_info)

Detailed Description

DAC driver structure. General DAC functions implemented at the HAL layer follow this API.

Field Documentation

◆ close

ssp_err_t(* dac_api_t::close) (dac_ctrl_t *p_ctrl)

Close the D/A Converter.

Implemented as

- R_DAC_Close()
- R_DAC8_Close()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ infoGet

ssp_err_t(* dac_api_t::infoGet) (dac_info_t *const p_info)

Get information about DAC Resolution and store it in provided pointer p_info.

Implemented as

- R_DAC_InfoGet()
- R_DAC8_InfoGet()

Parameters

[out]	p_info	Collection of information for this DAC.

◆ open

ssp_err_t(* dac_api_t::open) (dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_DAC_Open()
- R_DAC8_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

ssp_err_t(* dac_api_t::start) (dac_ctrl_t *p_ctrl)

Start the D/A Converter if it has not been started yet.

Implemented as

- R_DAC_Start()
- R_DAC8_Start()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ stop

ssp_err_t(* dac_api_t::stop) (dac_ctrl_t *p_ctrl)

Stop the D/A Converter if the converter is running.

Implemented as

- R_DAC_Stop()
- R_DAC8_Stop()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ versionGet

ssp_err_t(* dac_api_t::versionGet) (ssp_version_t *p_version)

Get version and store it in provided pointer p_version.

Implemented as

- R_DAC_VersionGet()
- R_DAC8_VersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ write

ssp_err_t(* dac_api_t::write) (dac_ctrl_t *p_ctrl, dac_size_t value)

Write sample value to the D/A Converter.

Implemented as

- R_DAC_Write()
- R_DAC8_Write()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.
[in]	value	Sample value to be written to the D/A Converter.

The documentation for this struct was generated from the following file:

- r_dac_api.h

dac_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DAC Interface

#include <r_dac_api.h>

Data Fields

dac_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
dac_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
dac_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_dac_api.h

5.1.4.11 Display Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for LCD panel displays. More...

Data Structures

struct	display_timing_t
struct	display_color_t
struct	display_coordinate_t
struct	display_brightness_t
struct	display_contrast_t
struct	display_correction_t
struct	gamma_correction_t
struct	display_gamma_correction_t
struct	display_clut_t
struct	display_input_cfg_t
struct	display_output_cfg_t
struct	display_layer_t
struct	display_callback_args_t
struct	display_cfg_t
struct	display_runtime_cfg_t

struct	display_clut_cfg_t
struct	display_status_t
struct	display_api_t
struct	display_instance_t

Typedefs

typedef void	display_ctrl_t

Enumerations

enum	display_frame_layer_t { DISPLAY_FRAME_LAYER_1 = 0, DISPLAY_FRAME_LAYER_2 = 1 }
enum	display_state_t { DISPLAY_STATE_CLOSED = 0, DISPLAY_STATE_OPENED = 1, DISPLAY_STATE_DISPLAYING = 2 }
enum	display_event_t { DISPLAY_EVENT_GR1_UNDERFLOW = 1, DISPLAY_EVENT_GR2_UNDERFLOW = 2, DISPLAY_EVENT_LINE_DETECTION = 3 }
enum	display_in_format_t { DISPLAY_IN_FORMAT_32BITS_ARGB8888 = 0, DISPLAY_IN_FORMAT_32BITS_RGB888 = 1, DISPLAY_IN_FORMAT_16BITS_RGB565 = 2, DISPLAY_IN_FORMAT_16BITS_ARGB1555 = 3, DISPLAY_IN_FORMAT_16BITS_ARGB4444 = 4, DISPLAY_IN_FORMAT_CLUT8 = 5, DISPLAY_IN_FORMAT_CLUT4 = 6, DISPLAY_IN_FORMAT_CLUT1 = 7 }
enum	display_out_format_t { DISPLAY_OUT_FORMAT_24BITS_RGB888, DISPLAY_OUT_FORMAT_18BITS_RGB666, DISPLAY_OUT_FORMAT_16BITS_RGB565, DISPLAY_OUT_FORMAT_8BITS_SERIAL }
enum	display_endian_t { DISPLAY_ENDIAN_LITTLE, DISPLAY_ENDIAN_BIG }
enum	display_color_order_t { DISPLAY_COLOR_ORDER_RGB, DISPLAY_COLOR_ORDER_BGR }
enum	display_signal_polarity_t { DISPLAY_SIGNAL_POLARITY_LOACTIVE, DISPLAY_SIGNAL_POLARITY_HIACTIVE }
enum	display_sync_edge_t { DISPLAY_SIGNAL_SYNC_EDGE_RISING, DISPLAY_SIGNAL_SYNC_EDGE_FALLING }
enum	display_fade_control_t { DISPLAY_FADE_CONTROL_NONE, DISPLAY_FADE_CONTROL_FADEIN,

	DISPLAY_FADE_CONTROL_FADEOUT }
enum	display_fade_status_t { DISPLAY_FADE_STATUS_NOT_UNDERWAY, DISPLAY_FADE_STATUS_FADING_UNDERWAY, DISPLAY_FADE_STATUS_UNCERTAIN }

Detailed Description

Interface for LCD panel displays.

Summary

The display interface provides standard display functionality:

- Signal timing configuration for LCD panels with RGB interface.
- Dot clock source selection (internal or external) and frequency divider.
- Blending of multiple graphics layers on the background screen.
- Color correction (brightness/configuration/gamma correction).
- Interrupts and callback function.

Implemented by: GLCDC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Display Interface description: Display Driver

Typedef Documentation

◆ display_ctrl_t

typedef void display_ctrl_t
Display control block. Allocate an instance specific control block to pass into the display API calls. **Implemented as** ◦ glcd_instance_ctrl_tDisplay control block

Enumeration Type Documentation

◆ display_color_order_t

enum display_color_order_t	
RGB color order select	
Enumerator	
DISPLAY_COLOR_ORDER_RGB	Color order RGB.
DISPLAY_COLOR_ORDER_BGR	Color order BGR.

◆ display_endian_t

enum display_endian_t	
Data endian select	
Enumerator	
DISPLAY_ENDIAN_LITTLE	Little-endian.
DISPLAY_ENDIAN_BIG	Big-endian.

◆ display_event_t

enum display_event_t	
Display event codes	
Enumerator	
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics frame1 underflow occurs.
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics frame2 underflow occurs.
DISPLAY_EVENT_LINE_DETECTION	Designated line is processed.

◆ display_fade_control_t

enum display_fade_control_t	
Fading control	
Enumerator	
DISPLAY_FADE_CONTROL_NONE	Applying no fading control.
DISPLAY_FADE_CONTROL_FADEIN	Applying fade-in control.
DISPLAY_FADE_CONTROL_FADEOUT	Applying fade-out control.

◆ display_fade_status_t

enum display_fade_status_t	
Fading status	
Enumerator	
DISPLAY_FADE_STATUS_NOT_UNDERWAY	Fade-in/fade-out is not in progress.
DISPLAY_FADE_STATUS_FADING_UNDERWAY	Fade-in or fade-out is in progress.
DISPLAY_FADE_STATUS_UNCERTAIN	Fade-in/fade-out status is uncertain just before hardware working.

◆ display_frame_layer_t

enum display_frame_layer_t	
Display frame number	
Enumerator	
DISPLAY_FRAME_LAYER_1	Frame layer 1.
DISPLAY_FRAME_LAYER_2	Frame layer 2.

◆ display_in_format_t

enum display_in_format_t	
Input format setting	
Enumerator	
DISPLAY_IN_FORMAT_32BITS_ARGB8888	ARGB8888, 32 bits.
DISPLAY_IN_FORMAT_32BITS_RGB888	RGB888, 32 bits.
DISPLAY_IN_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB1555	ARGB1555, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB4444	ARGB4444, 16 bits.
DISPLAY_IN_FORMAT_CLUT8	CLUT8.
DISPLAY_IN_FORMAT_CLUT4	CLUT4.
DISPLAY_IN_FORMAT_CLUT1	CLUT1.

◆ display_out_format_t

enum display_out_format_t	
Output format setting	
Enumerator	
DISPLAY_OUT_FORMAT_24BITS_RGB888	RGB888, 24 bits.
DISPLAY_OUT_FORMAT_18BITS_RGB666	RGB666, 18 bits.
DISPLAY_OUT_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_OUT_FORMAT_8BITS_SERIAL	SERIAL, 8 bits.

◆ display_signal_polarity_t

enum display_signal_polarity_t	
Polarity of a signal select	
Enumerator	
DISPLAY_SIGNAL_POLARITY_LOACTIVE	Low active signal.
DISPLAY_SIGNAL_POLARITY_HIACTIVE	High active signal.

◆ display_state_t

enum display_state_t	
Display interface operation state	
Enumerator	
DISPLAY_STATE_CLOSED	Display closed.
DISPLAY_STATE_OPENED	Display opened.
DISPLAY_STATE_DISPLAYING	Displaying.

◆ display_sync_edge_t

enum display_sync_edge_t	
Signal synchronization edge select	
Enumerator	
DISPLAY_SIGNAL_SYNC_EDGE_RISING	Signal is synchronized to rising edge.
DISPLAY_SIGNAL_SYNC_EDGE_FALLING	Signal is synchronized to falling edge.

display_timing_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

uint16_t	total_cyc Total cycles in one line or total lines in one frame.
uint16_t	display_cyc Active video cycles or lines.
uint16_t	back_porch Back poach cycles or lines.
uint16_t	sync_width Sync signal asserting width.
display_signal_polarity_t	sync_polarity Sync signal polarity.

Detailed Description

Display signal timing setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_color_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Detailed Description

RGB Color setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_coordinate_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

int16_t	x Coordinate X, this allows to set signed value.
int16_t	y Coordinate Y, this allows to set signed value.

Detailed Description

Contrast (gain) correction setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_brightness_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

bool	enable Brightness Correction On/Off.
uint16_t	r Brightness (DC) adjustment for R channel.
uint16_t	g

	Brightness (DC) adjustment for G channel.
uint16_t	b Brightness (DC) adjustment for B channel.

Detailed Description

Brightness (DC) correction setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_contrast_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

bool	enable Contrast Correction On/Off.
uint8_t	r Contrast (gain) adjustment for R channel.
uint8_t	g Contrast (gain) adjustment for G channel.
uint8_t	b Contrast (gain) adjustment for B channel.

Detailed Description

Contrast (gain) correction setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_correction_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_brightness_t	brightness Brightness.
display_contrast_t	contrast Contrast.

Detailed Description

Color correction setting

The documentation for this struct was generated from the following file:

- r_display_api.h

gamma_correction_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

bool	enable Gamma Correction On/Off.

uint16_t	gain [DISPLAY_GAMMA_CURVE_ELEMENT_NUM] Gain adjustment.
uint16_t	threshold [DISPLAY_GAMMA_CURVE_ELEMENT_NUM] Start threshold.

Detailed Description

Gamma correction setting for each color

The documentation for this struct was generated from the following file:

- r_display_api.h

display_gamma_correction_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

gamma_correction_t	r Gamma correction for R channel.
gamma_correction_t	g Gamma correction for G channel.
gamma_correction_t	b Gamma correction for B channel.

Detailed Description

Gamma correction setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_clut_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

uint32_t	color_num The number of colors in CLUT.
const uint32_t *	p_clut Address of the area storing the CLUT data (in ARGB8888 format)

Detailed Description

CLUT setting

The documentation for this struct was generated from the following file:

- r_display_api.h

display_input_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

uint32_t *	p_base Base address to the frame buffer.
uint16_t	hsize Horizontal pixel size in a line.

uint16_t	vsize Vertical pixel size in a frame.
uint32_t	hstride Memory stride (bytes) in a line.
display_in_format_t	format Input format setting.
bool	line_descending_enable Line descending enable.
bool	lines_repeat_enable Line repeat enable.
uint16_t	lines_repeat_times Expected number of line repeating.

Detailed Description

Graphics plane input configuration structure

The documentation for this struct was generated from the following file:

- r_display_api.h

display_output_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_timing_t	htiming

	Horizontal display cycle setting.
display_timing_t	vtiming Vertical display cycle setting.
display_out_format_t	format Output format setting.
display_endian_t	endian Bit order of output data.
display_color_order_t	color_order Color order in pixel.
display_signal_polarity_t	data_enable_polarity Data Enable signal polarity.
display_sync_edge_t	sync_edge Signal sync edge selection.
display_color_t	bg_color Background color.
display_brightness_t	brightness Brightness setting.
display_contrast_t	contrast Contrast setting.
display_gamma_correction_t *	p_gamma_correction Pointer to gamma correction setting.
bool	dithering_on

	Dithering on/off.

Detailed Description

Display output configuration structure

The documentation for this struct was generated from the following file:

- r_display_api.h

display_layer_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_coordinate_t	coordinate Blending location (starting point of image)
display_color_t	bg_color Color outside region.
display_fade_control_t	fade_control Layer fade-in/out control on/off.
uint8_t	fade_speed Layer fade-in/out frame rate.

Detailed Description

Graphics layer blend setup parameter structure

The documentation for this struct was generated from the following file:

- r_display_api.h

display_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_event_t	event Event code.
void const *	p_context Context provided to user during callback.

Detailed Description

Display callback parameter definition

The documentation for this struct was generated from the following file:

- r_display_api.h

display_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_input_cfg_t	input [DISPLAY_FRAME_LAYER_2+1] Graphics input frame setting. More...
display_output_cfg_t	output Graphics output frame setting.
display_layer_t	layer [DISPLAY_FRAME_LAYER_2+1]

	Graphics layer blend setting.
uint8_t	line_detect_ipl Line detect interrupt priority.
uint8_t	underflow_1_ipl Underflow 1 interrupt priority.
uint8_t	underflow_2_ipl Underflow 1 interrupt priority.
void(*	p_callback)(display_callback_args_t *p_args) Pointer to callback function. More...
void const *	p_context User defined context passed into callback function.
void const *	p_extend Display hardware dependent configuration. More...

Detailed Description

Display main configuration structure

Field Documentation

◆ input

display_input_cfg_t display_cfg_t::input[DISPLAY_FRAME_LAYER_2+1]
Graphics input frame setting. Generic configuration for display devices

◆ p_callback

void(* display_cfg_t::p_callback) (display_callback_args_t *p_args)
Pointer to callback function. Configuration for display event processing

◆ p_extend

void const* display_cfg_t::p_extend
Display hardware dependent configuration. Pointer to display peripheral specific configuration

---

The documentation for this struct was generated from the following file:

- r_display_api.h

display_runtime_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_input_cfg_t	input Graphics input frame setting. More...
display_layer_t	layer Graphics layer alpha blending setting.

Detailed Description

Display main configuration structure

Field Documentation

◆ input

display_input_cfg_t display_runtime_cfg_t::input
Graphics input frame setting. Generic configuration for display devices

---

The documentation for this struct was generated from the following file:

- r_display_api.h

display_clut_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

uint32_t *	p_base Pointer to CLUT source data.
uint16_t	start Beginning of CLUT entry to be updated.
uint16_t	size Size of CLUT entry to be updated.

Detailed Description

Display CLUT configuration structure

The documentation for this struct was generated from the following file:

- r_display_api.h

display_status_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_state_t	state Status of GLCD module.
display_fade_status_t	fade_status [DISPLAY_FRAME_LAYER_2+1] Status of fade-in/fade-out status.

Detailed Description

Display Status

The documentation for this struct was generated from the following file:

- r_display_api.h

display_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

ssp_err_t(*	open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)
ssp_err_t(*	close)(display_ctrl_t *const p_ctrl)
ssp_err_t(*	start)(display_ctrl_t *const p_ctrl)
ssp_err_t(*	stop)(display_ctrl_t *const p_ctrl)
ssp_err_t(*	layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)

ssp_err_t(*	correction)(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
ssp_err_t(*	clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
ssp_err_t(*	statusGet)(display_ctrl_t const *const p_ctrl, display_status_t *const p_status)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

Shared Interface definition for display peripheral

Field Documentation

◆ close

ssp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)

Close display device.

Implemented as

- R_GLCD_Close()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ clut

ssp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)

Set CLUT for display device.

Implemented as

- R_GLCD_ClutUpdate()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_clut_cfg	Pointer to CLUT configuration structure.
[in]	frame	Number of frame buffer corresponding to the CLUT.

◆ correction

ssp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)

Color correction.

Implemented as

- R_GLCD_ColorCorrection()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	param	Pointer to color correction configuration structure.

◆ layerChange

ssp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)

Change layer parameters at runtime.

Implemented as

- R_GLCD_LayerChange()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_cfg	Pointer to run-time layer configuration structure.
[in]	frame	Number of graphic frames.

◆ open

ssp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)

Open display device.

Implemented as

- R_GLCD_Open()

Parameters

[in,out]	p_ctrl	Pointer to display interface control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to display configuration structure. All elements of this structure must be set by user.

◆ start

ssp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)

Display start.

Implemented as

- R_GLCD_Start()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ statusGet

ssp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status)

Get status for display device.

Implemented as

- R_GLCD_StatusGet()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	status	Pointer to display interface status structure.

◆ stop

ssp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)

Display stop.

Implemented as

- R_GLCD_Stop()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ versionGet

ssp_err_t(* display_api_t::versionGet) (ssp_version_t *p_version)
Get version. **Implemented as** ◦ R_GLCD_VersionGet() **Parameters**

[in]	p_version	Pointer to the memory to store the version information.

The documentation for this struct was generated from the following file:

- r_display_api.h

display_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Display Interface

#include <r_display_api.h>

Data Fields

display_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
display_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
display_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_display_api.h

5.1.4.12 DOC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for the Data Operation Circuit. More...

Data Structures

struct	doc_callback_args_t
struct	doc_data_t
struct	doc_cfg_t
struct	doc_api_t
struct	doc_instance_t

Macros

#define	DOC_API_VERSION_MAJOR (2U)

Typedefs

typedef uint16_t	doc_size_t
typedef void	doc_ctrl_t

Enumerations

enum	doc_event_t { DOC_EVENT_COMPARISON_MISMATCH = 0x00, DOC_EVENT_ADDITION = 0x01, DOC_EVENT_SUBTRACTION = 0x02, DOC_EVENT_COMPARISON_MATCH = 0x04 }
enum	doc_status_t { DOC_STATUS_CONDITION_FALSE = 0, DOC_STATUS_CONDITION_TRUE = 1 }

Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Implemented by: DOC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

DOC Interface description: Data Operation Circuit Driver

Macro Definition Documentation

◆ DOC_API_VERSION_MAJOR

#define DOC_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ doc_ctrl_t

typedef void doc_ctrl_t
DOC control block. Allocate an instance specific control block to pass into the DOC API calls. **Implemented as** ◦ doc_instance_ctrl_t

◆ doc_size_t

typedef uint16_t doc_size_t
Size of the comparison data supported by the Data Operation Circuit (DOC)

Enumeration Type Documentation

◆ doc_event_t

enum doc_event_t	
Event that can trigger a callback function.	
Enumerator	
DOC_EVENT_COMPARISON_MISMATCH	Comparison of data has resulted in a mismatch.
DOC_EVENT_ADDITION	Addition of data has resulted in a value greater than H'FFFF.
DOC_EVENT_SUBTRACTION	Subtraction of data has resulted in a value less than H'0000.
DOC_EVENT_COMPARISON_MATCH	Comparison of data has resulted in a match.

◆ doc_status_t

enum doc_status_t	
Status of the data comparison operation.	
Enumerator	
DOC_STATUS_CONDITION_FALSE	Data comparison condition NOT met (match or mismatch), addition result NOT > H'FFFF, subtraction result NOT < H'0000.
DOC_STATUS_CONDITION_TRUE	Data comparison condition met (match or mismatch), addition result > H'FFFF, subtraction result < H'0000.

doc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DOC Interface

#include <r_doc_api.h>

Data Fields

doc_event_t	event The event is used to identify what caused the callback.

void const *	p_context

Detailed Description

Callback function parameter data.

Field Documentation

◆ p_context

void const* doc_callback_args_t::p_context
Placeholder for user data. Set in doc_api_t::open function in doc_cfg_t.

The documentation for this struct was generated from the following file:

- r_doc_api.h

doc_data_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DOC Interface

#include <r_doc_api.h>

Data Fields

doc_size_t	dodir Value to be written to the DOC DODIR.
doc_size_t	dodsr Value to be written to the DOC DODSR.

Detailed Description

Data to be written to DOC register for comparison/addition/subtraction.

The documentation for this struct was generated from the following file:

- r_doc_api.h

doc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DOC Interface

#include <r_doc_api.h>

Data Fields

doc_event_t	event Select enumerated value from doc_event_t.
uint8_t	irq_ipl DOC interrupt priority.
void(*	p_callback)(doc_callback_args_t *p_args)
void const *	p_context

Detailed Description

User configuration structure, used in the open function.

Field Documentation

◆ p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)
Callback provided when a DOC ISR occurs.

◆ p_context

void const* doc_cfg_t::p_context
Placeholder for user data. Passed to the user callback in doc_callback_args_t.

The documentation for this struct was generated from the following file:

- r_doc_api.h

doc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DOC Interface

#include <r_doc_api.h>

Data Fields

ssp_err_t(*	open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
ssp_err_t(*	close)(doc_ctrl_t *const p_ctrl)
ssp_err_t(*	statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
ssp_err_t(*	statusClear)(doc_ctrl_t *const p_ctrl)
ssp_err_t(*	write)(doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
ssp_err_t(*	inputRegisterWrite)(doc_ctrl_t *const p_ctrl, doc_size_t data)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Allow the driver to be reconfigured. Will reduce power consumption.

Implemented as

- R_DOC_Close()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.

◆ inputRegisterWrite

ssp_err_t(* doc_api_t::inputRegisterWrite) (doc_ctrl_t *const p_ctrl, doc_size_t data)

Write to the DODIR register.

Implemented as

- R_DOC_InputRegisterWrite()

Precondition

Call doc_api_t::open to configure the DOC before using this function.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	data	Data to be written to DOC DODIR register.

◆ open

ssp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_DOC_Open()

Precondition

Peripheral clocks should be configured prior to calling this function.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ statusClear

ssp_err_t(* doc_api_t::statusClear) (doc_ctrl_t *const p_ctrl)

Clear DOPCF status flag.

Implemented as

- R_DOC_StatusClear()

Precondition

Call doc_api_t::open to configure the DOC before using this function.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.

◆ statusGet

ssp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t *p_status)

Get the DOC status and stores it in the provided pointer p_status.

Implemented as

- R_DOC_StatusGet()

Precondition

Call doc_api_t::open to configure the DOC before using this function.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[out]	p_status	Indicates the status of the comparison/addition/subtraction operation. Result will be one of doc_status_t.

◆ versionGet

ssp_err_t(* doc_api_t::versionGet) (ssp_version_t *const p_version)

Get version and stores it in provided pointer p_version.

Implemented as

- R_DOC_VersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ write

ssp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)

Write to the DODIR and DODSR registers.

Implemented as

- R_DOC_Write()

Precondition

Call doc_api_t::open to configure the DOC before using this function.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	p_data	Pointer to data to be written to DOC DODIR and DODSR registers.

The documentation for this struct was generated from the following file:

- r_doc_api.h

doc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » DOC Interface

#include <r_doc_api.h>

Data Fields

doc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
doc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
doc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_doc_api.h

5.1.4.13 events and peripheral definitions

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for the Event Link Controller. More...

Data Structures

struct	elc_link_t
struct	elc_cfg_t
struct	elc_api_t
struct	elc_instance_t

Enumerations

enum	elc_software_event_t { ELC_SOFTWARE_EVENT_0, ELC_SOFTWARE_EVENT_1 }

Detailed Description

Interface for the Event Link Controller.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Event Link Controller Interface description: ELC Driver

Related SSP architecture topics:

- What is an SSP Interface? SSP Interfaces
- What is a SSP Layer? SSP Predefined Layers
- How to use SSP Interfaces and Modules? Using SSP Modules

Event Link Controller Interface description: ELC Driver

Enumeration Type Documentation

◆ elc_software_event_t

enum elc_software_event_t	
Software event number	
Enumerator	
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.

elc_link_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » events and peripheral definitions

#include <r_elc_api.h>

Data Fields

elc_peripheral_t	peripheral Peripheral to receive the signal.
elc_event_t	event Signal that gets sent to the Peripheral.

Detailed Description

Individual event link. The actual peripheral definitions can be found in the MCU specific (ie. /mcu/S124/bsp_elc.h) bsp_elc.h files.

The documentation for this struct was generated from the following file:

- r_elc_api.h

elc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » events and peripheral definitions

#include <r_elc_api.h>

Data Fields

bool	autostart Start operation and enable interrupts during open().
uint32_t	link_count Number of event links.
elc_link_t const *	link_list Event links.

Detailed Description

Main configuration structure for the Event Link Controller

The documentation for this struct was generated from the following file:

- r_elc_api.h

elc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » events and peripheral definitions

#include <r_elc_api.h>

Data Fields

ssp_err_t(*	init)(elc_cfg_t const *const p_cfg)
ssp_err_t(*	softwareEventGenerate)(elc_software_event_t event_num)
ssp_err_t(*	linkSet)(elc_peripheral_t peripheral, elc_event_t signal)
ssp_err_t(*	linkBreak)(elc_peripheral_t peripheral)
ssp_err_t(*	enable)(void)
ssp_err_t(*	disable)(void)

```
ssp_err_t(*  versionGet )(ssp_version_t *const p_version)
```

Detailed Description

ELC driver structure. General ELC functions implemented at the HAL layer follow this API.

Field Documentation

◆ disable

ssp_err_t(* elc_api_t::disable) (void)

Disable the operation of the Event Link Controller.

Implemented as

- R_ELC_Disable()

◆ enable

ssp_err_t(* elc_api_t::enable) (void)

Enable the operation of the Event Link Controller.

Implemented as

- R_ELC_Enable()

◆ init

ssp_err_t(* elc_api_t::init) (elc_cfg_t const *const p_cfg)

Initialize all links in the Event Link Controller.

Implemented as

- R_ELC_Init()

Parameters

[in]	p_cfg	Pointer to configuration structure.

◆ linkBreak

ssp_err_t(* elc_api_t::linkBreak) (elc_peripheral_t peripheral)

Break an event link.

Implemented as

- R_ELC_LinkBreak()

Parameters

[in]	peripheral	The peripheral that should no longer be linked.

◆ linkSet

ssp_err_t(* elc_api_t::linkSet) (elc_peripheral_t peripheral, elc_event_t signal)

Create a single event link.

Implemented as

- R_ELC_LinkSet()

Parameters

[in]	peripheral	The peripheral block that will receive the event signal.
[in]	signal	The event signal.

◆ softwareEventGenerate

ssp_err_t(* elc_api_t::softwareEventGenerate) (elc_software_event_t event_num)

Generate a software event in the Event Link Controller.

Implemented as

- R_ELC_SoftwareEventGenerate()

Parameters

[in]	eventNum	Software event number to be generated.

◆ versionGet

ssp_err_t(* elc_api_t::versionGet) (ssp_version_t *const p_version)
Get the driver version based on compile time macros. **Implemented as** ◦ R_ELC_VersionGet() **Parameters** [out] p_version is value returned.

The documentation for this struct was generated from the following file:

- r_elc_api.h

elc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » events and peripheral definitions

#include <r_elc_api.h>

Data Fields

elc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
elc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_elc_api.h

5.1.4.14 External IRQ Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for detecting external interrupts. More...

Data Structures

struct	external_irq_callback_args_t
struct	external_irq_cfg_t
struct	external_irq_api_t
struct	external_irq_instance_t

Macros

#define	EXTERNAL_IRQ_API_VERSION_MAJOR (2U) EXTERNAL IRQ API version number (Major)
#define	EXTERNAL_IRQ_API_VERSION_MINOR (0U) EXTERNAL IRQ API version number (Minor)

Typedefs

typedef void	external_irq_ctrl_t

Enumerations

enum	external_irq_trigger_t { EXTERNAL_IRQ_TRIG_FALLING = 0, EXTERNAL_IRQ_TRIG_RISING = 1, EXTERNAL_IRQ_TRIG_BOTH_EDGE = 2, EXTERNAL_IRQ_TRIG_LEVEL_LOW = 3 }
enum	external_irq_pclk_div_t { EXTERNAL_IRQ_PCLK_DIV_BY_1 = 0, EXTERNAL_IRQ_PCLK_DIV_BY_8 = 1, EXTERNAL_IRQ_PCLK_DIV_BY_32 = 2, EXTERNAL_IRQ_PCLK_DIV_BY_64 = 3 }

Detailed Description

Interface for detecting external interrupts.

Summary

The external IRQ interface supports external inputs, for example input from pins or capacitive touch buttons. When an input trigger is detected, a user provided callback function will be called.

Implemented by: ICU

Related interfaces: Key Matrix Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

External IRQ Interface description: External IRQ Driver

Typedef Documentation

◆ external_irq_ctrl_t

typedef void external_irq_ctrl_t
External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. **Implemented as** ◦ icu_instance_ctrl_t

Enumeration Type Documentation

◆ external_irq_pclk_div_t

enum external_irq_pclk_div_t	
External IRQ input pin digital filtering sample clock divisor settings.	
Enumerator	
EXTERNAL_IRQ_PCLK_DIV_BY_1	Filter using PCLK divided by 1.
EXTERNAL_IRQ_PCLK_DIV_BY_8	Filter using PCLK divided by 8.
EXTERNAL_IRQ_PCLK_DIV_BY_32	Filter using PCLK divided by 32.
EXTERNAL_IRQ_PCLK_DIV_BY_64	Filter using PCLK divided by 64.

◆ external_irq_trigger_t

enum external_irq_trigger_t	
Trigger type: rising edge, falling edge, both edges, low level.	
Enumerator	
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.

external_irq_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » External IRQ Interface

#include <r_external_irq_api.h>

Data Fields

void const *	p_context
uint32_t	channel The physical hardware channel that caused the interrupt.

Detailed Description

Callback function parameter data

Field Documentation

◆ p_context

void const* external_irq_callback_args_t::p_context
Placeholder for user data. Set in external_irq_api_t::open function in external_irq_cfg_t.

The documentation for this struct was generated from the following file:

- r_external_irq_api.h

external_irq_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » External IRQ Interface

#include <r_external_irq_api.h>

Data Fields

uint8_t	channel Hardware channel used.
uint8_t	irq_ipl Interrupt priority.
external_irq_trigger_t	trigger Trigger setting.
external_irq_pclk_div_t	pclk_div Digital filter clock divisor setting.
bool	autostart Start operation and enable interrupts during open().
bool	filter_enable Digital filter enable/disable setting.
void(*	p_callback)(external_irq_callback_args_t *p_args)
void const *	p_context
void const *	p_extend External IRQ hardware dependent configuration.

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ p_callback

void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)
Callback provided external input trigger occurs.

◆ p_context

void const* external_irq_cfg_t::p_context
Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

The documentation for this struct was generated from the following file:

- r_external_irq_api.h

external_irq_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » External IRQ Interface

#include <r_external_irq_api.h>

Data Fields

ssp_err_t(*	open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
ssp_err_t(*	enable)(external_irq_ctrl_t *const p_ctrl)
ssp_err_t(*	disable)(external_irq_ctrl_t *const p_ctrl)
ssp_err_t(*	triggerSet)(external_irq_ctrl_t *const p_ctrl, external_irq_trigger_t const trigger)
ssp_err_t(*	filterEnable)(external_irq_ctrl_t *const p_ctrl)
ssp_err_t(*	filterDisable)(external_irq_ctrl_t *const p_ctrl)
ssp_err_t(*	close)(external_irq_ctrl_t *const p_ctrl)

ssp_err_t(* versionGet)(ssp_version_t *const p_version)

Detailed Description

External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

- R_ICU_ExternalIrqClose()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ disable

ssp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)

Disable callback when external IRQ occurs.

Implemented as

- R_ICU_ExternalIrqDisable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ enable

ssp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)

Enable callback when external IRQ occurs.

Implemented as

- R_ICU_ExternalIrqEnable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ filterDisable

ssp_err_t(* external_irq_api_t::filterDisable) (external_irq_ctrl_t *const p_ctrl)

Disable noise filter.

Implemented as

- R_ICU_ExternalIrqFilterDisable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ filterEnable

ssp_err_t(* external_irq_api_t::filterEnable) (external_irq_ctrl_t *const p_ctrl)

Enables noise filter.

Implemented as

- R_ICU_ExternalIrqFilterEnable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ open

ssp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_ICU_ExternalIrqOpen()

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ triggerSet

ssp_err_t(* external_irq_api_t::triggerSet) (external_irq_ctrl_t *const p_ctrl, external_irq_trigger_t const trigger)

Set trigger.

Implemented as

- R_ICU_ExternalIrqTriggerSet()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.
[in]	trigger	Trigger type

◆ versionGet

ssp_err_t(* external_irq_api_t::versionGet) (ssp_version_t *const p_version)
Get version and store it in provided pointer p_version. **Implemented as** ◦ R_ICU_ExternalIrqVersionGet() **Parameters** [out] p_version Code and API version used.

The documentation for this struct was generated from the following file:

- r_external_irq_api.h

external_irq_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » External IRQ Interface

#include <r_external_irq_api.h>

Data Fields

external_irq_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
external_irq_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
external_irq_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_external_irq_api.h

5.1.4.15 Flash Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for the flash controller. More...

Data Structures

struct	flash_fmi_block_info_t
struct	flash_fmi_regions_t
struct	flash_info_t
struct	flash_callback_args_t
struct	flash_cfg_t
struct	flash_api_t
struct	flash_instance_t

Macros

#define	FLASH_API_VERSION_MAJOR (2U)
#define	FLASH_API_VERSION_MINOR (0U)

Typedefs

typedef void	flash_ctrl_t

Enumerations

enum	flash_result_t { FLASH_RESULT_BLANK, FLASH_RESULT_NOT_BLANK, FLASH_RESULT_BGO_ACTIVE }
enum	flash_startup_area_swap_t { FLASH_STARTUP_AREA_BLOCK1 = 0, FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BTFLG }
enum	flash_event_t { FLASH_EVENT_ERASE_COMPLETE, FLASH_EVENT_WRITE_COMPLETE, FLASH_EVENT_BLANK, FLASH_EVENT_NOT_BLANK, FLASH_EVENT_ERR_DF_ACCESS, FLASH_EVENT_ERR_CF_ACCESS, FLASH_EVENT_ERR_CMD_LOCKED, FLASH_EVENT_ERR_FAILURE,

	FLASH_EVENT_ERR_ONE_BIT }
enum	flash_id_code_mode_t { FLASH_ID_CODE_MODE_UNLOCKED, FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT, FLASH_ID_CODE_MODE_LOCKED }

Detailed Description

Interface for the flash controller.

Summary

The Flash interface provides the functionality necessary to read, write, erase and blank check the Flash memory. Additionally functions are provided to configure the access window and swap areas of the flash memory.

Implemented by:

- High-performance Flash
- Low Power Flash

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Flash Interface description: Flash Driver

Macro Definition Documentation

◆ FLASH_API_VERSION_MAJOR

#define FLASH_API_VERSION_MAJOR (2U)
FLASH HAL API version number (Major)

◆ FLASH_API_VERSION_MINOR

#define FLASH_API_VERSION_MINOR (0U)
FLASH HAL API version number (Minor)

Typedef Documentation

◆ flash_ctrl_t

typedef void flash_ctrl_t

Flash control block. Allocate an instance specific control block to pass into the flash API calls.

Implemented as

- flash_lp_instance_ctrl_t
- flash_hp_instance_ctrl_t

Enumeration Type Documentation

◆ flash_event_t

enum flash_event_t

Event types returned by the ISR callback when used in Data Flash BGO mode

Enumerator	
FLASH_EVENT_ERASE_COMPLETE	Erase operation successfully completed.
FLASH_EVENT_WRITE_COMPLETE	Write operation successfully completed.
FLASH_EVENT_BLANK	Blank check operation successfully completed. Specified area is blank.
FLASH_EVENT_NOT_BLANK	Blank check operation successfully completed. Specified area is NOT blank.
FLASH_EVENT_ERR_DF_ACCESS	Data Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CF_ACCESS	Code Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CMD_LOCKED	Operation failed, FCU is in Locked state (often result of an illegal command)
FLASH_EVENT_ERR_FAILURE	Erase or Program Operation failed.
FLASH_EVENT_ERR_ONE_BIT	A 1-bit error has been corrected when reading the flash memory area by the sequencer.

◆ flash_id_code_mode_t

enum flash_id_code_mode_t	
ID Code Modes for writing to ID code registers	
Enumerator	
FLASH_ID_CODE_MODE_UNLOCKED	ID code is ignored.
FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT	ID code is checked. All erase is available.
FLASH_ID_CODE_MODE_LOCKED	ID code is checked.

◆ flash_result_t

enum flash_result_t	
Result type for certain operations	
Enumerator	
FLASH_RESULT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_NOT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_BGO_ACTIVE	Flash is configured for BGO mode. Result is returned in callback.

◆ flash_startup_area_swap_t

enum flash_startup_area_swap_t	
Parameter for specifying the startup area swap being requested by startupAreaSelect()	
Enumerator	
FLASH_STARTUP_AREA_BLOCK1	Startup area will be set to Block 1.
FLASH_STARTUP_AREA_BLOCK0	Startup area will be set to Block 0.
FLASH_STARTUP_AREA_BTFLG	Startup area will be set based on the value of the BTFLG.

flash_fmi_block_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

uint32_t	block_section_st_addr starting address for this block section (blocks of this size)
uint32_t	block_section_end_addr ending address for this block section (blocks of this size)
uint32_t	block_size Flash erase block size.
uint32_t	block_size_write Flash write block size.

Detailed Description

Flash block details stored in factory flash.

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_fmi_regions_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

uint32_t	num_regions Length of block info array.
flash_fmi_block_info_t const	p_block_array

*	Block info array base address.

Detailed Description

Flash block details

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

flash_fmi_regions_t	code_flash Information about the code flash regions.
flash_fmi_regions_t	data_flash Information about the code flash regions.

Detailed Description

Information about the flash blocks

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

flash_event_t	event Event can be used to identify what caused the callback (flash ready or error).
void const *	p_context Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

bool	data_flash_bgo True if BGO (Background Operation) is enabled for Data Flash.
void(*	p_callback)(flash_callback_args_t *p_args) Callback provided when a Flash interrupt ISR occurs.
void const *	p_extend FLASH hardware dependent configuration.
void const *	p_context

	Placeholder for user data. Passed to user callback in flash_callback_args_t.
uint8_t	irq_ipl Flash ready interrupt priority.
uint8_t	err_irq_ipl Flash error interrupt priority (unused in r_flash_lp)

Detailed Description

FLASH Configuration

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

ssp_err_t(*	open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
ssp_err_t(*	write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
ssp_err_t(*	read)(flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)
ssp_err_t(*	erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)
ssp_err_t(*	blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
ssp_err_t(*	infoGet)(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

ssp_err_t(*	close)(flash_ctrl_t *const p_ctrl)
ssp_err_t(*	statusGet)(flash_ctrl_t *const p_ctrl)
ssp_err_t(*	accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
ssp_err_t(*	accessWindowClear)(flash_ctrl_t *const p_ctrl)
ssp_err_t(*	idCodeSet)(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)
ssp_err_t(*	reset)(flash_ctrl_t *const p_ctrl)
ssp_err_t(*	updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)
ssp_err_t(*	startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

Shared Interface definition for FLASH

Field Documentation

◆ accessWindowClear

ssp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)

Clear any existing Code Flash access window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowClear()
- R_FLASH_HP_AccessWindowClear()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window.

◆ accessWindowSet

ssp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

Set Access Window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowSet()
- R_FLASH_HP_AccessWindowSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window.

◆ blankCheck

ssp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)

Blank check FLASH device.

Implemented as

- R_FLASH_LP_BlankCheck()
- R_FLASH_HP_BlankCheck()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	address	The starting address of the Flash area to blank check.
[in]	num_bytes	Specifies the number of bytes that need to be checked. See the specific handler for details.
[out]	p_blank_check_result	Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result.

◆ close

ssp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)

Close FLASH device.

Implemented as

- R_FLASH_LP_Close()
- R_FLASH_HP_Close()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ erase

ssp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Erase FLASH device.

Implemented as

R_FLASH_LP_Erase() R_FLASH_HP_Erase()

Parameters

[in]	p_ctrl	Control for the FLASH device.
[in]	address	The block containing this address is the first block erased.
[in]	num_blocks	Specifies the number of blocks to be erased, the starting block determined by the block_erase_address.

◆ idCodeSet

ssp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode. e.g. uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0 and with mode FLASH_ID_CODE_MODE_LOCKED will result in an ID code of 00112233445566778899aabbccddee80

Implemented as

- R_FLASH_LP_IdCodeSet()
- R_FLASH_HP_IdCodeSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	p_id_bytes	Ponter to the ID Code to be written.
[in]	mode	Mode used for checking the ID code.

◆ infoGet

ssp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

Close FLASH device.

Implemented as

- R_FLASH_LP_InfoGet()
- R_FLASH_HP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_info	Pointer to FLASH info structure.

◆ open

ssp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

Open FLASH device.

Implemented as

- R_FLASH_LP_Open()
- R_FLASH_HP_Open()

Parameters

[out]	p_ctrl	Pointer to FLASH device control. Must be declared by user. Value set here.
[in]	flash_cfg_t	Pointer to FLASH configuration structure. All elements of this structure must be set by the user.

◆ read

ssp_err_t(* flash_api_t::read) (flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)

Read FLASH device.

Implemented as

- R_FLASH_LP_Read()
- R_FLASH_HP_Read()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	p_dest_address	Pointer to caller's destination buffer used to hold the data read from Flash.
[in]	flash_address	Code Flash or Data Flash starting address to read from.
[in]	num_bytes	The number of bytes to read.

◆ reset

ssp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)

Reset function for FLASH device.

Implemented as

- R_FLASH_LP_Reset()
- R_FLASH_HP_Reset()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ startupAreaSelect

ssp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block.

Implemented as

- R_FLASH_LP_StartUpAreaSelect()
- R_FLASH_HP_StartUpAreaSelect()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	swap_type	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
[in]	is_temporary	True or false. See table below.

swap_type | is_temporary | Operation FLASH_STARTUP_AREA_BLOCK0: false On next reset Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK0 | false | On next reset Startup area will be Block 0. Block 0.
FLASH_STARTUP_AREA_BLOCK1: false On next reset Startup area will be Block 1.
FLASH_STARTUP_AREA_BLOCK1 | true | Startup area is immediately, but temporarily switched to Block 1. Block 1.
FLASH_STARTUP_AREA_BTFLG | true | Startup area is immediately, but temporarily switched to... taken.
the Block determined by the Configuration BTFLG.

◆ statusGet

ssp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl)

Get Status for FLASH device.

Implemented as

- R_FLASH_LP_StatusGet()
- R_FLASH_HP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ updateFlashClockFreq

ssp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)

Update Flash clock frequency (FCLK) and recalculate timeout values

Implemented as

- R_FLASH_LP_UpdateFlashClockFreq()
- R_FLASH_HP_UpdateFlashClockFreq()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ versionGet

ssp_err_t(* flash_api_t::versionGet) (ssp_version_t *p_version)

Get Flash driver version.

Implemented as

- R_FLASH_LP_VersionGet()
- R_FLASH_HP_VersionGet()

Parameters

[out]	p_version	Returns version.

◆ write

ssp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write FLASH device.

Implemented as

- R_FLASH_LP_Write()
- R_FLASH_HP_Write()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	src_address	Address of the buffer containing the data to write to Flash.
[in]	flash_address	Code Flash or Data Flash address to write. The address must be on a programming line boundary.
[in]	num_bytes	The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF.

Warning

Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_BYTES being returned and no data written.

The documentation for this struct was generated from the following file:

- r_flash_api.h

flash_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Flash Interface

#include <r_flash_api.h>

Data Fields

flash_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
flash_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
flash_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_flash_api.h

5.1.4.16 FMI Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for reading on-chip factory information. More...

Data Structures

struct	fmi_api_t
struct	fmi_instance_t

Macros

#define	FMI_API_VERSION_MAJOR (2U)

Detailed Description

Interface for reading on-chip factory information.

Summary

The FMI (Factory MCU Information) module provides a function for reading the Product Information record.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

FMI Interface description: FMI Driver

Macro Definition Documentation

◆ FMI_API_VERSION_MAJOR

#define FMI_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

fmi_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » FMI Interface

#include <r_fmi_api.h>

Data Fields

ssp_err_t(*	init)(void)
ssp_err_t(*	productInfoGet)(fmi_product_info_t **pp_product_info)
ssp_err_t(*	uniqueIdGet)(fmi_unique_id_t *p_unique_id)
ssp_err_t(*	productFeatureGet)(ssp_feature_t const *const p_feature, fmi_feature_info_t *const p_info)
ssp_err_t(*	eventInfoGet)(ssp_feature_t const *const p_feature, ssp_signal_t signal, fmi_event_info_t *const p_info)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

fmi driver structure. General fmi functions implemented at the HAL layer will follow this API.

Field Documentation

◆ eventInfoGet

ssp_err_t(* fmi_api_t::eventInfoGet) (ssp_feature_t const *const p_feature, ssp_signal_t signal, fmi_event_info_t *const p_info)

Get event information and store it in p_info.

Parameters

[in]	p_feature	Definition of SSP feature.
[in]	signal	Feature signal ID.
[out]	p_info	Event information for feature signal.

Implemented as

- R_FMI_EventInfoGet()

◆ init

ssp_err_t(* fmi_api_t::init) (void)

Initialize the FMI base pointer.

Implemented as

- R_FMI_Init()

◆ productFeatureGet

ssp_err_t(* fmi_api_t::productFeatureGet) (ssp_feature_t const *const p_feature, fmi_feature_info_t *const p_info)

Get feature information and store it in p_info.

Parameters

[in]	p_feature	Definition of SSP feature.
[out]	p_info	Feature specific information.

Implemented as

- R_FMI_ProductFeatureGet()

◆ productInfoGet

ssp_err_t(* fmi_api_t::productInfoGet) (fmi_product_info_t **pp_product_info)

Get product information record address into caller's pointer.

Warning

fmi_product_info_t::unique_id is deprecated and will not contain a unique ID if the factory MCU information is linked in by the application code. Use fmi_api_t::uniqueIdGet for the unique ID.

Parameters

[in,out]	pp_product_info	Pointer to store pointer to product info.

Implemented as

- R_FMI_ProductInfoGet()

◆ uniqueIdGet

ssp_err_t(* fmi_api_t::uniqueIdGet) (fmi_unique_id_t *p_unique_id)

Copy unique ID into p_unique_id.

Parameters

[out]	p_unique_id	Pointer to unique ID.

Implemented as

- R_FMI_UniqueIdGet()

◆ versionGet

ssp_err_t(* fmi_api_t::versionGet) (ssp_version_t *const p_version)

Get the driver version based on compile time macros.

Implemented as

- R_FMI_VersionGet()

The documentation for this struct was generated from the following file:

- r_fmi_api.h

fmi_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » FMI Interface

#include <r_fmi_api.h>

Data Fields

fmi_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_fmi_api.h

5.1.4.17 I2C Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for I2C communication. More...

Data Structures

struct	i2c_callback_args_t
struct	i2c_cfg_t
struct	i2c_api_master_t
struct	i2c_api_slave_t
struct	i2c_master_instance_t
struct	i2c_slave_instance_t

Typedefs

typedef void	i2c_ctrl_t

Enumerations

enum	i2c_rate_t { I2C_RATE_STANDARD = 100000, I2C_RATE_FAST = 400000, I2C_RATE_FASTPLUS = 1000000 }
enum	i2c_addr_mode_t { I2C_ADDR_MODE_7BIT = 1, I2C_ADDR_MODE_10BIT }
enum	i2c_event_t { I2C_EVENT_ABORTED = 1, I2C_EVENT_RX_COMPLETE = 2, I2C_EVENT_TX_COMPLETE = 3, I2C_SLAVE_EVENT_RX_REQUEST = 4, I2C_SLAVE_EVENT_TX_REQUEST = 5, I2C_SLAVE_EVENT_RX_MORE_REQUEST = 6, I2C_SLAVE_EVENT_TX_MORE_REQUEST = 7 }
enum	i2c_hw_err_event_t { I2C_HW_ERR_EVENT_TIMEOUT = 1, I2C_HW_ERR_EVENT_ARBITRATION_LOSS = 2, I2C_HW_ERR_EVENT_NACK = 16, I2C_HW_ERR_EVENT_UNDEFINED = 255 }

Detailed Description

Interface for I2C communication.

Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which can return an event code

Implemented by:

- Simple I2C on SCI
- IIC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

I2C Interface description: I2C Master Driver and I2C Slave Driver

Typedef Documentation

◆ i2c_ctrl_t

typedef void i2c_ctrl_t
I2C control block. Allocate an instance specific control block to pass into the I2C API calls. **Implemented as** ◦ sci_i2c_instance_ctrl_t ◦ riic_instance_ctrl_t

Enumeration Type Documentation

◆ i2c_addr_mode_t

enum i2c_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ i2c_event_t

enum i2c_event_t	
Callback events	
Enumerator	
I2C_EVENT_ABORTED	A transfer was aborted.
I2C_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_EVENT_TX_COMPLETE	A transmit operation was completed successfully.
I2C_SLAVE_EVENT_RX_REQUEST	A read operation expected from slave. Detected a write from master.
I2C_SLAVE_EVENT_TX_REQUEST	A write operation expected from slave. Detected a read from master.
I2C_SLAVE_EVENT_RX_MORE_REQUEST	A read operation expected from slave. Master sends out more data than configured to be read in slave.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	A write operation expected from slave. Master requests more data than configured to be written by slave.

◆ i2c_hw_err_event_t

enum i2c_hw_err_event_t	
RIIC master hardware error callback events	
Enumerator	
I2C_HW_ERR_EVENT_TIMEOUT	Timeout generated during transfer.
I2C_HW_ERR_EVENT_ARBITRATION_LOSS	Arbitration loss.
I2C_HW_ERR_EVENT_NACK	NACK event generation.
I2C_HW_ERR_EVENT_UNDEFINED	Error not defined.

◆ i2c_rate_t

enum i2c_rate_t	
Communication speed options	
Enumerator	
I2C_RATE_STANDARD	100 kHz
I2C_RATE_FAST	400 kHz
I2C_RATE_FASTPLUS	1 MHz

i2c_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

void const *const	p_context Pointer to user-provided context.
uint32_t const	bytes Number of received/transmitted bytes in buff.
i2c_event_t const	event Event code.
i2c_hw_err_event_t const	i2c_hw_err_event IIC Master hardware error events.

Detailed Description

I2C callback parameter definition

The documentation for this struct was generated from the following file:

- r_i2c_api.h

i2c_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

uint8_t	channel Identifier recognizable by implementation. More...
i2c_rate_t	rate Device's maximum clock rate from enum i2c_rate_t.
uint16_t	slave The address of the slave device.
i2c_addr_mode_t	addr_mode Indicates how slave fields should be interpreted.
uint16_t	sda_delay The SDA output delay.
uint8_t	rxi_ipl Receive interrupt priority.
uint8_t	txi_ipl Transmit interrupt priority.
uint8_t	tei_ipl Transmit end interrupt priority.

uint8_t	eri_ipl Error interrupt priority.
transfer_instance_t const *	p_transfer_tx DTC instance for I2C transmit.Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx DTC instance for I2C receive. Set to NULL if unused.
void(*	p_callback)(i2c_callback_args_t *p_args) Pointer to callback function. More...
void const *	p_context Pointer to the user-provided context.
void const *	p_extend Any configuration data needed by the hardware. More...

Detailed Description

I2C configuration block

Field Documentation

◆ channel

uint8_t i2c_cfg_t::channel
Identifier recognizable by implementation. Generic configuration

◆ p_callback

void(* i2c_cfg_t::p_callback) (i2c_callback_args_t *p_args)
Pointer to callback function. Parameters to control software behavior

◆ p_extend

void const* i2c_cfg_t::p_extend
Any configuration data needed by the hardware. Implementation-specific configuration

◆ p_transfer_tx

transfer_instance_t const* i2c_cfg_t::p_transfer_tx
DTC instance for I2C transmit.Set to NULL if unused. DTC/DMA support

---

The documentation for this struct was generated from the following file:

- r_i2c_api.h

i2c_api_master_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

ssp_err_t(*	open)(i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)
ssp_err_t(*	close)(i2c_ctrl_t *const p_ctrl)
ssp_err_t(*	read)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
ssp_err_t(*	write)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
ssp_err_t(*	reset)(i2c_ctrl_t *const p_ctrl)
ssp_err_t(*	slaveAddressSet)(i2c_ctrl_t *const p_ctrl, uint16_t const slave, i2c_addr_mode_t const addr_mode)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Interface definition for I2C access as master

Field Documentation

◆ close

ssp_err_t(* i2c_api_master_t::close) (i2c_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C device.

Implemented as

- R_RIIC_MasterClose()
- R_SCI_SIIC_MasterClose()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ open

ssp_err_t(* i2c_api_master_t::open) (i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)

Opens the I2C driver and initializes the hardware.

Implemented as

- R_RIIC_MasterOpen()
- R_SCI_SIIC_MasterOpen()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

ssp_err_t(* i2c_api_master_t::read) (i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C device.

Implemented as

- R_RIIC_MasterRead()
- R_SCI_SIIC_MasterRead()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.
[in]	restart	Specify if the restart condition should be issued after reading.

◆ reset

ssp_err_t(* i2c_api_master_t::reset) (i2c_ctrl_t *const p_ctrl)

Performs a reset of the peripheral.

Implemented as

- R_RIIC_MasterReset()
- R_SCI_SIIC_MasterReset()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ slaveAddressSet

ssp_err_t(* i2c_api_master_t::slaveAddressSet) (i2c_ctrl_t *const p_ctrl, uint16_t const slave, i2c_addr_mode_t const addr_mode)

Sets address of the slave device without reconfiguring the bus.

Implemented as

- R_RIIC_MasterSlaveAddressSet()
- R_SCI_SIIC_MasterSlaveAddressSet()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	slave_address	Address of the slave device.
[in]	address_mode	Addressing mode.

◆ versionGet

ssp_err_t(* i2c_api_master_t::versionGet) (ssp_version_t *const p_version)

Gets version information and stores it in the provided version struct.

Implemented as

- R_RIIC_MasterVersionGet()
- R_SCI_SIIC_MasterVersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ write

ssp_err_t(* i2c_api_master_t::write) (i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

Performs a write operation on an I2C device.

Implemented as

- R_RIIC_MasterWrite()
- R_SCI_SIIC_MasterWrite()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.
[in]	restart	Specify if the restart condition should be issued after writing.

The documentation for this struct was generated from the following file:

- r_i2c_api.h

i2c_api_slave_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

ssp_err_t(*	open)(i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)
ssp_err_t(*	close)(i2c_ctrl_t *const p_ctrl)
ssp_err_t(*	masterWriteSlaveRead)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
ssp_err_t(*	masterReadSlaveWrite)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

ssp_err_t(* versionGet)(ssp_version_t *const p_version)

Detailed Description

Interface definition for I2C access as slave

Field Documentation

◆ close

ssp_err_t(* i2c_api_slave_t::close) (i2c_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C device.

Implemented as

- R_RIIC_SlaveClose()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_slave_t::open call.

◆ masterReadSlaveWrite

ssp_err_t(* i2c_api_slave_t::masterReadSlaveWrite) (i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Performs a write operation on an I2C device.

Implemented as

- R_RIIC_MasterReadSlaveWrite()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_slave_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ masterWriteSlaveRead

ssp_err_t(* i2c_api_slave_t::masterWriteSlaveRead) (i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Performs a read operation on an I2C device.

Implemented as

- R_RIIC_MasterWriteSlaveRead()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_slave_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ open

ssp_err_t(* i2c_api_slave_t::open) (i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)

Opens the I2C driver and initializes the hardware.

Implemented as

- R_RIIC_SlaveOpen()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ versionGet

ssp_err_t(* i2c_api_slave_t::versionGet) (ssp_version_t *const p_version)

Gets version information and stores it in the provided version struct.

Implemented as

- R_RIIC_SlaveVersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_i2c_api.h

i2c_master_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

i2c_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
i2c_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
i2c_api_master_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_i2c_api.h

i2c_slave_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2C Interface

#include <r_i2c_api.h>

Data Fields

i2c_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
i2c_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
i2c_api_slave_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_i2c_api.h

5.1.4.18 I2S Interface

Renesas Synergy Software Package Reference » HAL Interfaces

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication. More...

Data Structures

struct	i2s_callback_args_t
struct	i2s_info_t
struct	i2s_cfg_t
struct	i2s_api_t
struct	i2s_instance_t

Macros

#define	I2S_API_VERSION_MAJOR (2U)

Typedefs

typedef void	i2s_ctrl_t

Enumerations

enum	i2s_pcm_width_t { I2S_PCM_WIDTH_8_BITS = 0, I2S_PCM_WIDTH_16_BITS = 1, I2S_PCM_WIDTH_18_BITS = 2, I2S_PCM_WIDTH_20_BITS = 3, I2S_PCM_WIDTH_22_BITS = 4, I2S_PCM_WIDTH_24_BITS = 5 }
enum	i2s_word_length_t { I2S_WORD_LENGTH_8_BITS = 0, I2S_WORD_LENGTH_16_BITS = 1, I2S_WORD_LENGTH_24_BITS = 2, I2S_WORD_LENGTH_32_BITS = 3 }
enum	i2s_event_t { I2S_EVENT_IDLE, I2S_EVENT_TX_EMPTY, I2S_EVENT_RX_FULL }
enum	i2s_dir_t { I2S_DIR_TX, I2S_DIR_RX, I2S_DIR_TX_RX }
enum	i2s_mode_t { I2S_MODE_SLAVE = 0, I2S_MODE_MASTER = 1 }
enum	i2s_mute_t { I2S_MUTE_ON = 0, I2S_MUTE_OFF = 1 }
enum	i2s_ws_continue_t { I2S_WS_CONTINUE_ON = 0, I2S_WS_CONTINUE_OFF = 1 }
enum	i2s_status_t { I2S_STATUS_IN_USE, I2S_STATUS_STOPPED }

Detailed Description

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

Known Implementations

SSI

Related modules

See also: I2S Audio Playback Framework

Macro Definition Documentation

◆ I2S_API_VERSION_MAJOR

#define I2S_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ i2s_ctrl_t

typedef void i2s_ctrl_t
I2S control block. Allocate an instance specific control block to pass into the I2S API calls. **Implemented as** ◦ ssi_instance_ctrl_t

Enumeration Type Documentation

◆ i2s_dir_t

enum i2s_dir_t	
I2S communication direction	
Enumerator	
I2S_DIR_TX	Transmit direction only.
I2S_DIR_RX	Receive direction only.
I2S_DIR_TX_RX	Transmit and receive direction.

◆ i2s_event_t

enum i2s_event_t	
Events that can trigger a callback function	
Enumerator	
I2S_EVENT_IDLE	Communication is idle.
I2S_EVENT_TX_EMPTY	Transmit buffer is below FIFO trigger level.
I2S_EVENT_RX_FULL	Receive buffer is above FIFO trigger level.

◆ i2s_mode_t

enum i2s_mode_t	
I2S communication mode	
Enumerator	
I2S_MODE_SLAVE	Slave mode.
I2S_MODE_MASTER	Master mode.

◆ i2s_mute_t

enum i2s_mute_t	
Mute audio samples.	
Enumerator	
I2S_MUTE_ON	Enable mute.
I2S_MUTE_OFF	Disable mute.

◆ i2s_pcm_width_t

enum i2s_pcm_width_t	
Audio PCM width	
Enumerator	
I2S_PCM_WIDTH_8_BITS	Using 8-bit PCM.
I2S_PCM_WIDTH_16_BITS	Using 16-bit PCM.
I2S_PCM_WIDTH_18_BITS	Using 18-bit PCM.
I2S_PCM_WIDTH_20_BITS	Using 20-bit PCM.
I2S_PCM_WIDTH_22_BITS	Using 22-bit PCM.
I2S_PCM_WIDTH_24_BITS	Using 24-bit PCM.

◆ i2s_status_t

enum i2s_status_t	
Possible status values returned by i2s_api_t::infoGet.	
Enumerator	
I2S_STATUS_IN_USE	I2S is in use.
I2S_STATUS_STOPPED	I2S is stopped.

◆ i2s_word_length_t

enum i2s_word_length_t	
Audio system word length.	
Enumerator	
I2S_WORD_LENGTH_8_BITS	Using 8-bit system word length.
I2S_WORD_LENGTH_16_BITS	Using 16-bit system word length.
I2S_WORD_LENGTH_24_BITS	Using 24-bit system word length.
I2S_WORD_LENGTH_32_BITS	Using 32-bit system word length.

◆ i2s_ws_continue_t

enum i2s_ws_continue_t	
Whether to continue WS (word select line) transmission during idle state.	
Enumerator	
I2S_WS_CONTINUE_ON	Enable WS continue mode.
I2S_WS_CONTINUE_OFF	Disable WS continue mode.

i2s_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2S Interface

#include <r_i2s_api.h>

Data Fields

void const *	p_context
i2s_event_t	event The event can be used to identify what caused the callback (overflow or error).

Detailed Description

Callback function parameter data

Field Documentation

◆ p_context

void const* i2s_callback_args_t::p_context
Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t.

The documentation for this struct was generated from the following file:

- r_i2s_api.h

i2s_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2S Interface

#include <r_i2s_api.h>

Data Fields

uint32_t	sampling_freq_hz Sampling frequency in Hertz.

Detailed Description

Timer information structure to store various information for an I2S instance.

The documentation for this struct was generated from the following file:

- r_i2s_api.h

i2s_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2S Interface

#include <r_i2s_api.h>

Data Fields

uint8_t	channel
i2s_pcm_width_t	pcm_width Audio PCM data width.
i2s_word_length_t	word_length Audio word length, bits must be >= i2s_cfg_t::pcm_width bits.
i2s_ws_continue_t	ws_continue Whether to continue WS transmission during idle state.
uint32_t	sampling_freq_hz Sampling frequency in Hertz.
i2s_mode_t	operating_mode Operating mode selection (i.e., Master/Slave mode)
uint32_t	audio_clk_freq_hz
timer_instance_t const *	p_timer
transfer_instance_t const *	p_transfer_tx
transfer_instance_t const *	p_transfer_rx
void(*	p_callback)(i2s_callback_args_t *p_args)

void const *	p_context
void const *	p_extend Extension parameter for hardware specific settings.
uint8_t	rxi_ipl Receive interrupt priority.
uint8_t	txi_ipl Transmit interrupt priority.
uint8_t	idle_err_ipl Idle/Error interrupt priority.

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ audio_clk_freq_hz

uint32_t i2s_cfg_t::audio_clk_freq_hz
Audio clock frequency in Hertz. Must be a multiple between 1 and 128 of (16 * i2s_cfg_t::sampling_freq_hz * (i2s_cfg_t::word_length <enum_value> + 1)

◆ channel

uint8_t i2s_cfg_t::channel
Select a channel corresponding to the channel number of the hardware.

◆ p_callback

void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)
Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void const* i2s_cfg_t::p_context
Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

◆ p_timer

timer_instance_t const* i2s_cfg_t::p_timer
To generate audio clock with GPT, link a timer instance here. Set to NULL if unused.

◆ p_transfer_rx

transfer_instance_t const* i2s_cfg_t::p_transfer_rx
To use DTC during read, link a DTC instance here. Set to NULL if unused.

◆ p_transfer_tx

transfer_instance_t const* i2s_cfg_t::p_transfer_tx
To use DTC during write, link a DTC instance here. Set to NULL if unused.

The documentation for this struct was generated from the following file:

- r_i2s_api.h

i2s_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2S Interface

#include <r_i2s_api.h>

Data Fields

ssp_err_t(*	open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
ssp_err_t(*	stop)(i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)
ssp_err_t(*	mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

ssp_err_t(*	write)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)
ssp_err_t(*	read)(i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)
ssp_err_t(*	writeRead)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)
ssp_err_t(*	infoGet)(i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)
ssp_err_t(*	close)(i2s_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

I2S functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_SSI_Close()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ infoGet

ssp_err_t(* i2s_api_t::infoGet) (i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)

Get instance specific information and store it in provided pointer p_info.

Implemented as

- R_SSI_InfoGet()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[out]	p_info	Collection of information for this instance.

◆ mute

ssp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

Enable or disable mute.

Implemented as

- R_SSI_Mute()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	mute_enable	Whether to enable or disable mute.

◆ open

ssp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_SSI_Open()

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call i2s_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ read

ssp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

Implemented as

- R_SSI_Read()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

◆ stop

ssp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)

Stop communication. Transmission is stopped when callback is called with I2S_EVENT_IDLE. Reception is stopped when callback is called with I2S_EVENT_RX_EMPTY.

Implemented as

- R_SSI_Stop()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	dir	Direction of communication to stop.

◆ versionGet

ssp_err_t(* i2s_api_t::versionGet) (ssp_version_t *const p_version)

Get version and store it in provided pointer p_version.

Implemented as

- R_SSI_VersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ write

ssp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.

Implemented as

- R_SSI_Write()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8.

◆ writeRead

ssp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

Implemented as

- R_SSI_WriteRead()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes.

The documentation for this struct was generated from the following file:

- r_i2s_api.h

i2s_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I2S Interface

#include <r_i2s_api.h>

Data Fields

i2s_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

i2s_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
i2s_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_i2s_api.h

5.1.4.19 Input Capture Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for sampling input signals for pulse width. More...

Data Structures

struct	input_capture_callback_args_t
struct	input_capture_capture_t
struct	input_capture_info_t
struct	input_capture_cfg_t
struct	input_capture_api_t
struct	input_capture_instance_t

Macros

#define	info_capture_info_t input_capture_info_t

Typedefs

typedef void	input_capture_ctrl_t

Enumerations

enum	input_capture_mode_t { INPUT_CAPTURE_MODE_PULSE_WIDTH, INPUT_CAPTURE_MODE_PERIOD, INPUT_CAPTURE_MODE_PULSE_COUNT }
enum	input_capture_signal_edge_t { INPUT_CAPTURE_SIGNAL_EDGE_RISING, INPUT_CAPTURE_SIGNAL_EDGE_FALLING }
enum	input_capture_signal_level_t { INPUT_CAPTURE_SIGNAL_LEVEL_NONE , INPUT_CAPTURE_SIGNAL_LEVEL_LOW, INPUT_CAPTURE_SIGNAL_LEVEL_HIGH }
enum	input_capture_repetition_t { INPUT_CAPTURE_REPETITION_PERIODIC, INPUT_CAPTURE_REPETITION_ONE_SHOT }
enum	input_capture_event_t { INPUT_CAPTURE_EVENT_MEASUREMENT, INPUT_CAPTURE_EVENT_OVERFLOW }
enum	input_capture_status_t { INPUT_CAPTURE_STATUS_IDLE, INPUT_CAPTURE_STATUS_CAPTURING }
enum	input_capture_variant_t { INPUT_CAPTURE_VARIANT_32_BIT, INPUT_CAPTURE_VARIANT_16_BIT }

Detailed Description

Interface for sampling input signals for pulse width.

Summary

The input capture interface provides for sampling of input signals to determine the width of a pulse (from one edge to the opposite edge). An interrupt can be triggered after each measurement is captured.

Implemented by:

- GPT Input Capture
- AGT Input Capture

See also: Timer Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Macro Definition Documentation

◆ info_capture_info_t

#define info_capture_info_t input_capture_info_t
Mapping of deprecated info_capture_info_t.

Typedef Documentation

◆ input_capture_ctrl_t

typedef void input_capture_ctrl_t
Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. **Implemented as** ◦ gpt_input_capture_instance_ctrl_t ◦ agt_input_capture_instance_ctrl_t

Enumeration Type Documentation

◆ input_capture_event_t

enum input_capture_event_t	
Events that can trigger a callback function	
Enumerator	
INPUT_CAPTURE_EVENT_MEASUREMENT	A capture measurement has been captured.
INPUT_CAPTURE_EVENT_OVERFLOW	A capture measurement overflowed the counter.

◆ input_capture_mode_t

enum input_capture_mode_t	
Input capture operational modes	
Enumerator	
INPUT_CAPTURE_MODE_PULSE_WIDTH	Measure a signal pulse width.
INPUT_CAPTURE_MODE_PERIOD	Measure a signal Cycle period.
INPUT_CAPTURE_MODE_PULSE_COUNT	Measure a signal event count.

◆ input_capture_repetition_t

enum input_capture_repetition_t	
Specifies either a one-time or continuous measurements	
Enumerator	
INPUT_CAPTURE_REPETITION_PERIODIC	Capture continuous measurements, until explicitly stopped or disabled.
INPUT_CAPTURE_REPETITION_ONE_SHOT	Capture a single measurement, then interrupts are disabled.

◆ input_capture_signal_edge_t

enum input_capture_signal_edge_t	
Input capture signal edge trigger	
Enumerator	
INPUT_CAPTURE_SIGNAL_EDGE_RISING	The capture begins at the rising edge.
INPUT_CAPTURE_SIGNAL_EDGE_FALLING	The capture begins at the falling edge.

◆ input_capture_signal_level_t

enum input_capture_signal_level_t	
Input capture signal level, primarily used for the enable signal	
Enumerator	
INPUT_CAPTURE_SIGNAL_LEVEL_NONE	Use this if signal_level is not applicable to a particular measurement.
INPUT_CAPTURE_SIGNAL_LEVEL_LOW	The capture is enabled at the low level.
INPUT_CAPTURE_SIGNAL_LEVEL_HIGH	The capture is enabled at the high level.

◆ input_capture_status_t

enum input_capture_status_t	
Input capture status.	
Enumerator	
INPUT_CAPTURE_STATUS_IDLE	The input capture timer is idle.
INPUT_CAPTURE_STATUS_CAPTURING	A capture measurement is in progress.

◆ input_capture_variant_t

enum input_capture_variant_t	
Input capture timer variant types.	
Enumerator	
INPUT_CAPTURE_VARIANT_32_BIT	32-bit timer
INPUT_CAPTURE_VARIANT_16_BIT	16-bit timer

input_capture_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

uint8_t	channel The channel being used.
input_capture_event_t	event The event that caused the interrupt and callback.
uint32_t	counter The value of the timer captured at the time of interrupt.
uint32_t	overflows

	The number of counter overflows that occurred during this measurement.
void const *	p_context Placeholder for user data, set in input_capture_cfg_t::p_context.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

input_capture_capture_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

uint32_t	counter The value of the timer captured at the time of interrupt.
uint32_t	overflows The number of counter overflows that occurred during this measurement.

Detailed Description

Capture data

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

input_capture_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

input_capture_status_t	status Whether or not a capture is in progress.
input_capture_variant_t	variant Whether timer is 16 or 32 bits.

Detailed Description

Driver information

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

input_capture_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

uint8_t	channel The channel in use.
uint8_t	capture_irq_ipl Capture interrupt priority.
uint8_t	overflow_irq_ipl Overflow interrupt priority.

input_capture_mode_t	mode The mode of measurement to be performed.
input_capture_signal_edge_t	edge The triggering edge to start a measurement (rise or fall).
input_capture_repetition_t	repetition One-shot or periodic measurement.
bool	autostart Specifies whether interrupts are enabled or not after open.
void const *	p_extend
void(*	p_callback)(input_capture_callback_args_t *p_args)
void const *	p_context Pointer to user's context data, to be passed to the callback.

Detailed Description

User configuration structure, passed to input_capture_api_t::open function

Field Documentation

◆ p_callback

void(* input_capture_cfg_t::p_callback) (input_capture_callback_args_t *p_args)
Pointer to user's callback function, or NULL if no interrupt desired.

◆ p_extend

void const* input_capture_cfg_t::p_extend
REQUIRED. Pointer to peripheral-specific extension parameters. See gpt_input_capture_extend_t for GPT.

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

input_capture_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

ssp_err_t(*	open)(input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)
ssp_err_t(*	disable)(input_capture_ctrl_t const *const p_ctrl)
ssp_err_t(*	enable)(input_capture_ctrl_t const *const p_ctrl)
ssp_err_t(*	infoGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)
ssp_err_t(*	lastCaptureGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)
ssp_err_t(*	close)(input_capture_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Input capture API structure. Functions implemented at the HAL layer will implement this API.

Field Documentation

◆ close

ssp_err_t(* input_capture_api_t::close) (input_capture_ctrl_t *const p_ctrl)

Close the input capture operation. Allows driver to be reconfigured, and may reduce power consumption.

Implemented as

- R_GPT_InputCaptureClose()
- R_AGT_InputCaptureClose()

Parameters

[in]	p_ctrl	Pointer to control block initialized by input_capture_api_t::open call.

◆ disable

ssp_err_t(* input_capture_api_t::disable) (input_capture_ctrl_t const *const p_ctrl)

Disables input capture measurement.

Implemented as

- R_GPT_InputCaptureDisable()
- R_AGT_InputCaptureDisable()

Parameters

[in]	p_ctrl	Pointer to control block initialized by input_capture_api_t::open call.

◆ enable

ssp_err_t(* input_capture_api_t::enable) (input_capture_ctrl_t const *const p_ctrl)

Enables input capture measurement.

Implemented as

- R_GPT_InputCaptureEnable()
- R_AGT_InputCaptureEnable()

Parameters

[in]	p_ctrl	Pointer to control block initialized by input_capture_api_t::open call.

Note

Interrupts may already be enabled if specified by input_capture_cfg_t::irq_enable.

◆ infoGet

ssp_err_t(* input_capture_api_t::infoGet) (input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)

Gets the status (running or not) of the measurement counter.

Implemented as

- R_GPT_InputCaptureInfoGet()
- R_AGT_InputCaptureInfoGet()

Parameters

[in]	p_ctrl	Pointer to control block initialized by input_capture_api_t::open call.
[out]	p_info	Pointer to returned status. Result will be one of input_capture_status_t.

◆ lastCaptureGet

ssp_err_t(* input_capture_api_t::lastCaptureGet) (input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)

Gets the last captured timer/counter value

Implemented as

- R_GPT_InputCaptureLastCaptureGet()
- R_AGT_InputCaptureLastCaptureGet()

Parameters

[in]	p_ctrl	Pointer to control block initialized by input_capture_api_t::open call.
[out]	p_counter	Pointer to location to store last captured counter value.

◆ open

ssp_err_t(* input_capture_api_t::open) (input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_GPT_InputCaptureOpen()
- R_AGT_InputCaptureOpen()

Note

To reconfigure after calling this function, call input_capture_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block: memory allocated by caller, contents filled in by open.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ versionGet

ssp_err_t(* input_capture_api_t::versionGet) (ssp_version_t *const p_version)
Gets the version of this API and stores it in structure pointed to by p_version. **Implemented as** ◦ R_GPT_InputCaptureVersionGet() ◦ R_AGT_InputCaptureVersionGet() **Parameters** [out] p_version Code and API version used.

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

input_capture_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Input Capture Interface

#include <r_input_capture_api.h>

Data Fields

input_capture_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
input_capture_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
input_capture_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_input_capture_api.h

5.1.4.20 I/O Port Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for accessing I/O ports and configuring I/O functionality. More...

Data Structures

struct	ioport_pin_cfg_t
struct	ioport_cfg_t
struct	ioport_api_t
struct	ioport_instance_t

Typedefs

typedef uint16_t	ioport_size_t IO port size on this device. More...

Enumerations

enum	ioport_level_t { IOPORT_LEVEL_LOW = 0, IOPORT_LEVEL_HIGH }
enum	ioport_direction_t { IOPORT_DIRECTION_INPUT = 0, IOPORT_DIRECTION_OUTPUT }
enum	ioport_port_t { IOPORT_PORT_00 = 0x0000, IOPORT_PORT_01 = 0x0100, IOPORT_PORT_02 = 0x0200, IOPORT_PORT_03 = 0x0300, IOPORT_PORT_04 = 0x0400, IOPORT_PORT_05 = 0x0500, IOPORT_PORT_06 = 0x0600, IOPORT_PORT_07 = 0x0700, IOPORT_PORT_08 = 0x0800, IOPORT_PORT_09 = 0x0900, IOPORT_PORT_10 = 0x0A00, IOPORT_PORT_11 = 0x0B00 }
enum	ioport_port_pin_t { IOPORT_PORT_00_PIN_00 = 0x0000, IOPORT_PORT_00_PIN_01 = 0x0001, IOPORT_PORT_00_PIN_02 = 0x0002, IOPORT_PORT_00_PIN_03 = 0x0003, IOPORT_PORT_00_PIN_04 = 0x0004, IOPORT_PORT_00_PIN_05 = 0x0005, IOPORT_PORT_00_PIN_06 = 0x0006, IOPORT_PORT_00_PIN_07 = 0x0007,

```
  IOPORT_PORT_00_PIN_08 = 0x0008, IOPORT_PORT_00_PIN_09 =
0x0009, IOPORT_PORT_00_PIN_10 = 0x000A,
IOPORT_PORT_00_PIN_11 = 0x000B,
  IOPORT_PORT_00_PIN_12 = 0x000C, IOPORT_PORT_00_PIN_13 =
0x000D, IOPORT_PORT_00_PIN_14 = 0x000E,
IOPORT_PORT_00_PIN_15 = 0x000F,
  IOPORT_PORT_01_PIN_00 = 0x0100, IOPORT_PORT_01_PIN_01 =
0x0101, IOPORT_PORT_01_PIN_02 = 0x0102,
IOPORT_PORT_01_PIN_03 = 0x0103,
  IOPORT_PORT_01_PIN_04 = 0x0104, IOPORT_PORT_01_PIN_05 =
0x0105, IOPORT_PORT_01_PIN_06 = 0x0106,
IOPORT_PORT_01_PIN_07 = 0x0107,
  IOPORT_PORT_01_PIN_08 = 0x0108, IOPORT_PORT_01_PIN_09 =
0x0109, IOPORT_PORT_01_PIN_10 = 0x010A,
IOPORT_PORT_01_PIN_11 = 0x010B,
  IOPORT_PORT_01_PIN_12 = 0x010C, IOPORT_PORT_01_PIN_13 =
0x010D, IOPORT_PORT_01_PIN_14 = 0x010E,
IOPORT_PORT_01_PIN_15 = 0x010F,
  IOPORT_PORT_02_PIN_00 = 0x0200, IOPORT_PORT_02_PIN_01 =
0x0201, IOPORT_PORT_02_PIN_02 = 0x0202,
IOPORT_PORT_02_PIN_03 = 0x0203,
  IOPORT_PORT_02_PIN_04 = 0x0204, IOPORT_PORT_02_PIN_05 =
0x0205, IOPORT_PORT_02_PIN_06 = 0x0206,
IOPORT_PORT_02_PIN_07 = 0x0207,
  IOPORT_PORT_02_PIN_08 = 0x0208, IOPORT_PORT_02_PIN_09 =
0x0209, IOPORT_PORT_02_PIN_10 = 0x020A,
IOPORT_PORT_02_PIN_11 = 0x020B,
  IOPORT_PORT_02_PIN_12 = 0x020C, IOPORT_PORT_02_PIN_13 =
0x020D, IOPORT_PORT_02_PIN_14 = 0x020E,
IOPORT_PORT_02_PIN_15 = 0x020F,
  IOPORT_PORT_03_PIN_00 = 0x0300, IOPORT_PORT_03_PIN_01 =
0x0301, IOPORT_PORT_03_PIN_02 = 0x0302,
IOPORT_PORT_03_PIN_03 = 0x0303,
  IOPORT_PORT_03_PIN_04 = 0x0304, IOPORT_PORT_03_PIN_05 =
0x0305, IOPORT_PORT_03_PIN_06 = 0x0306,
IOPORT_PORT_03_PIN_07 = 0x0307,
  IOPORT_PORT_03_PIN_08 = 0x0308, IOPORT_PORT_03_PIN_09 =
0x0309, IOPORT_PORT_03_PIN_10 = 0x030A,
IOPORT_PORT_03_PIN_11 = 0x030B,
  IOPORT_PORT_03_PIN_12 = 0x030C, IOPORT_PORT_03_PIN_13 =
0x030D, IOPORT_PORT_03_PIN_14 = 0x030E,
IOPORT_PORT_03_PIN_15 = 0x030F,
  IOPORT_PORT_04_PIN_00 = 0x0400, IOPORT_PORT_04_PIN_01 =
0x0401, IOPORT_PORT_04_PIN_02 = 0x0402,
IOPORT_PORT_04_PIN_03 = 0x0403,
  IOPORT_PORT_04_PIN_04 = 0x0404, IOPORT_PORT_04_PIN_05 =
0x0405, IOPORT_PORT_04_PIN_06 = 0x0406,
IOPORT_PORT_04_PIN_07 = 0x0407,
  IOPORT_PORT_04_PIN_08 = 0x0408, IOPORT_PORT_04_PIN_09 =
0x0409, IOPORT_PORT_04_PIN_10 = 0x040A,
IOPORT_PORT_04_PIN_11 = 0x040B,
  IOPORT_PORT_04_PIN_12 = 0x040C, IOPORT_PORT_04_PIN_13 =
0x040D, IOPORT_PORT_04_PIN_14 = 0x040E,
IOPORT_PORT_04_PIN_15 = 0x040F,
  IOPORT_PORT_05_PIN_00 = 0x0500, IOPORT_PORT_05_PIN_01 =
```

```
0x0501, IOPORT_PORT_05_PIN_02 = 0x0502,
IOPORT_PORT_05_PIN_03 = 0x0503,
  IOPORT_PORT_05_PIN_04 = 0x0504, IOPORT_PORT_05_PIN_05 =
0x0505, IOPORT_PORT_05_PIN_06 = 0x0506,
IOPORT_PORT_05_PIN_07 = 0x0507,
  IOPORT_PORT_05_PIN_08 = 0x0508, IOPORT_PORT_05_PIN_09 =
0x0509, IOPORT_PORT_05_PIN_10 = 0x050A,
IOPORT_PORT_05_PIN_11 = 0x050B,
  IOPORT_PORT_05_PIN_12 = 0x050C, IOPORT_PORT_05_PIN_13 =
0x050D, IOPORT_PORT_05_PIN_14 = 0x050E,
IOPORT_PORT_05_PIN_15 = 0x050F,
  IOPORT_PORT_06_PIN_00 = 0x0600, IOPORT_PORT_06_PIN_01 =
0x0601, IOPORT_PORT_06_PIN_02 = 0x0602,
IOPORT_PORT_06_PIN_03 = 0x0603,
  IOPORT_PORT_06_PIN_04 = 0x0604, IOPORT_PORT_06_PIN_05 =
0x0605, IOPORT_PORT_06_PIN_06 = 0x0606,
IOPORT_PORT_06_PIN_07 = 0x0607,
  IOPORT_PORT_06_PIN_08 = 0x0608, IOPORT_PORT_06_PIN_09 =
0x0609, IOPORT_PORT_06_PIN_10 = 0x060A,
IOPORT_PORT_06_PIN_11 = 0x060B,
  IOPORT_PORT_06_PIN_12 = 0x060C, IOPORT_PORT_06_PIN_13 =
0x060D, IOPORT_PORT_06_PIN_14 = 0x060E,
IOPORT_PORT_06_PIN_15 = 0x060F,
  IOPORT_PORT_07_PIN_00 = 0x0700, IOPORT_PORT_07_PIN_01 =
0x0701, IOPORT_PORT_07_PIN_02 = 0x0702,
IOPORT_PORT_07_PIN_03 = 0x0703,
  IOPORT_PORT_07_PIN_04 = 0x0704, IOPORT_PORT_07_PIN_05 =
0x0705, IOPORT_PORT_07_PIN_06 = 0x0706,
IOPORT_PORT_07_PIN_07 = 0x0707,
  IOPORT_PORT_07_PIN_08 = 0x0708, IOPORT_PORT_07_PIN_09 =
0x0709, IOPORT_PORT_07_PIN_10 = 0x070A,
IOPORT_PORT_07_PIN_11 = 0x070B,
  IOPORT_PORT_07_PIN_12 = 0x070C, IOPORT_PORT_07_PIN_13 =
0x070D, IOPORT_PORT_07_PIN_14 = 0x070E,
IOPORT_PORT_07_PIN_15 = 0x070F,
  IOPORT_PORT_08_PIN_00 = 0x0800, IOPORT_PORT_08_PIN_01 =
0x0801, IOPORT_PORT_08_PIN_02 = 0x0802,
IOPORT_PORT_08_PIN_03 = 0x0803,
  IOPORT_PORT_08_PIN_04 = 0x0804, IOPORT_PORT_08_PIN_05 =
0x0805, IOPORT_PORT_08_PIN_06 = 0x0806,
IOPORT_PORT_08_PIN_07 = 0x0807,
  IOPORT_PORT_08_PIN_08 = 0x0808, IOPORT_PORT_08_PIN_09 =
0x0809, IOPORT_PORT_08_PIN_10 = 0x080A,
IOPORT_PORT_08_PIN_11 = 0x080B,
  IOPORT_PORT_08_PIN_12 = 0x080C, IOPORT_PORT_08_PIN_13 =
0x080D, IOPORT_PORT_08_PIN_14 = 0x080E,
IOPORT_PORT_08_PIN_15 = 0x080F,
  IOPORT_PORT_09_PIN_00 = 0x0900, IOPORT_PORT_09_PIN_01 =
0x0901, IOPORT_PORT_09_PIN_02 = 0x0902,
IOPORT_PORT_09_PIN_03 = 0x0903,
  IOPORT_PORT_09_PIN_04 = 0x0904, IOPORT_PORT_09_PIN_05 =
0x0905, IOPORT_PORT_09_PIN_06 = 0x0906,
IOPORT_PORT_09_PIN_07 = 0x0907,
  IOPORT_PORT_09_PIN_08 = 0x0908, IOPORT_PORT_09_PIN_09 =
0x0909, IOPORT_PORT_09_PIN_10 = 0x090A,
```

```
IOPORT_PORT_09_PIN_11 = 0x090B,
  IOPORT_PORT_09_PIN_12 = 0x090C, IOPORT_PORT_09_PIN_13 =
0x090D, IOPORT_PORT_09_PIN_14 = 0x090E,
IOPORT_PORT_09_PIN_15 = 0x090F,
  IOPORT_PORT_10_PIN_00 = 0x0A00, IOPORT_PORT_10_PIN_01 =
0x0A01, IOPORT_PORT_10_PIN_02 = 0x0A02,
IOPORT_PORT_10_PIN_03 = 0x0A03,
  IOPORT_PORT_10_PIN_04 = 0x0A04, IOPORT_PORT_10_PIN_05 =
0x0A05, IOPORT_PORT_10_PIN_06 = 0x0A06,
IOPORT_PORT_10_PIN_07 = 0x0A07,
  IOPORT_PORT_10_PIN_08 = 0x0A08, IOPORT_PORT_10_PIN_09 =
0x0A09, IOPORT_PORT_10_PIN_10 = 0x0A0A,
IOPORT_PORT_10_PIN_11 = 0x0A0B,
  IOPORT_PORT_10_PIN_12 = 0x0A0C, IOPORT_PORT_10_PIN_13 =
0x0A0D, IOPORT_PORT_10_PIN_14 = 0x0A0E,
IOPORT_PORT_10_PIN_15 = 0x0A0F,
  IOPORT_PORT_11_PIN_00 = 0x0B00, IOPORT_PORT_11_PIN_01 =
0x0B01, IOPORT_PORT_11_PIN_02 = 0x0B02,
IOPORT_PORT_11_PIN_03 = 0x0B03,
  IOPORT_PORT_11_PIN_04 = 0x0B04, IOPORT_PORT_11_PIN_05 =
0x0B05, IOPORT_PORT_11_PIN_06 = 0x0B06,
IOPORT_PORT_11_PIN_07 = 0x0B07,
  IOPORT_PORT_11_PIN_08 = 0x0B08, IOPORT_PORT_11_PIN_09 =
0x0B09, IOPORT_PORT_11_PIN_10 = 0x0B0A,
IOPORT_PORT_11_PIN_11 = 0x0B0B,
  IOPORT_PORT_11_PIN_12 = 0x0B0C, IOPORT_PORT_11_PIN_13 =
0x0B0D, IOPORT_PORT_11_PIN_14 = 0x0B0E,
IOPORT_PORT_11_PIN_15 = 0x0B0F
}
```

enum

```
ioport_peripheral_t {
  IOPORT_PERIPHERAL_IO = 0x00, IOPORT_PERIPHERAL_DEBUG =
(0x00UL << IOPORT_PRV_PFS_PSEL_OFFSET),
IOPORT_PERIPHERAL_AGT = (0x01UL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_GPT0 =
(0x02UL << IOPORT_PRV_PFS_PSEL_OFFSET),
  IOPORT_PERIPHERAL_GPT1 = (0x03UL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_SCI0_2_4_6_8
= (0x04UL << IOPORT_PRV_PFS_PSEL_OFFSET),
IOPORT_PERIPHERAL_SCI1_3_5_7_9 = (0x05UL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_RSPI =
(0x06UL << IOPORT_PRV_PFS_PSEL_OFFSET),
  IOPORT_PERIPHERAL_RIIC = (0x07UL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_KEY =
(0x08UL << IOPORT_PRV_PFS_PSEL_OFFSET),
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC = (0x09UL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_CAC_AD =
(0x0AUL << IOPORT_PRV_PFS_PSEL_OFFSET),
  IOPORT_PERIPHERAL_BUS = (0x0BUL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_CTSU =
(0x0CUL << IOPORT_PRV_PFS_PSEL_OFFSET),
IOPORT_PERIPHERAL_LCDC = (0x0DUL <<
IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_DALI =
(0x0EUL << IOPORT_PRV_PFS_PSEL_OFFSET),
  IOPORT_PERIPHERAL_CAN = (0x10UL <<
```

	IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_QSPI = (0x11UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_SSI = (0x12UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_USB_FS = (0x13UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_USB_HS = (0x14UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_SDHI_MMC = (0x15UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_ETHER_MII = (0x16UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_ETHER_RMII = (0x17UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_PDC = (0x18UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_LCD_GRAPHICS = (0x19UL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_TRACE = (0x1AUL << IOPORT_PRV_PFS_PSEL_OFFSET), IOPORT_PERIPHERAL_END }
enum	ioport_ethernet_channel_t { IOPORT_ETHERNET_CHANNEL_0 = 0x10, IOPORT_ETHERNET_CHANNEL_1 = 0x20, IOPORT_ETHERNET_CHANNEL_END }
enum	ioport_ethernet_mode_t { IOPORT_ETHERNET_MODE_MII = 0, IOPORT_ETHERNET_MODE_RMII, IOPORT_ETHERNET_MODE_END }

enum	ioport_cfg_options_t {
	IOPORT_CFG_PORT_DIRECTION_INPUT = 0x00000000, IOPORT_CFG_PORT_DIRECTION_OUTPUT = 0x00000004, IOPORT_CFG_PORT_OUTPUT_LOW = 0x00000000, IOPORT_CFG_PORT_OUTPUT_HIGH = 0x00000001, IOPORT_CFG_PULLUP_ENABLE = 0x00000010, IOPORT_CFG_PIM_TTL = 0x00000020, IOPORT_CFG_NMOS_ENABLE = 0x00000040, IOPORT_CFG_PMOS_ENABLE = 0x00000080, IOPORT_CFG_DRIVE_MID = 0x00000400, IOPORT_CFG_DRIVE_MID_IIC = 0x00000C00, IOPORT_CFG_DRIVE_HIGH = 0x00000C00, IOPORT_CFG_EVENT_RISING_EDGE = 0x00001000, IOPORT_CFG_EVENT_FALLING_EDGE = 0x00002000, IOPORT_CFG_EVENT_BOTH_EDGES = 0x00003000, IOPORT_CFG_IRQ_ENABLE = 0x00004000, IOPORT_CFG_ANALOG_ENABLE = 0x00008000, IOPORT_CFG_PERIPHERAL_PIN = 0x00010000
	}

Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

IOPORT Interface description: I/O Port Driver

Typedef Documentation

◆ ioport_size_t

typedef uint16_t ioport_size_t
IO port size on this device. IO port type used with ports

Enumeration Type Documentation

◆ ioport_cfg_options_t

enum ioport_cfg_options_t	
Options to configure pin functions	
Enumerator	
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_PIM_TTL	Enables the pin's input mode.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_PMOS_ENABLE	Enables the pin's PMOS open-drain ouput.
IOPORT_CFG_DRIVE_MID	Sets pin drive output to medium.
IOPORT_CFG_DRIVE_MID_IIC	Sets pin to drive output needed for IIC on a 20mA port.
IOPORT_CFG_DRIVE_HIGH	Sets pin drive output to high.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.

◆ ioport_direction_t

enum ioport_direction_t	
Direction of individual pins	
Enumerator	
IOPORT_DIRECTION_INPUT	Input.
IOPORT_DIRECTION_OUTPUT	Output.

◆ ioport_ethernet_channel_t

enum ioport_ethernet_channel_t	
Superset of Ethernet channels.	
Enumerator	
IOPORT_ETHERNET_CHANNEL_0	Used to select Ethernet channel 0.
IOPORT_ETHERNET_CHANNEL_1	Used to select Ethernet channel 1.
IOPORT_ETHERNET_CHANNEL_END	Marks end of enum - used by parameter checking.

◆ ioport_ethernet_mode_t

enum ioport_ethernet_mode_t	
Superset of Ethernet PHY modes.	
Enumerator	
IOPORT_ETHERNET_MODE_MII	Ethernet PHY mode set to MII.
IOPORT_ETHERNET_MODE_RMII	Ethernet PHY mode set to RMII.
IOPORT_ETHERNET_MODE_END	Marks end of enum - used by parameter checking.

◆ ioport_level_t

enum ioport_level_t	
Levels that can be set and read for individual pins	
Enumerator	
IOPORT_LEVEL_LOW	Low.
IOPORT_LEVEL_HIGH	High.

◆ ioport_peripheral_t

enum ioport_peripheral_t	
Superset of all peripheral functions.	
Enumerator	
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin.
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin.
IOPORT_PERIPHERAL_AGT	Pin will function as an AGT.
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT.
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT.
IOPORT_PERIPHERAL_SCI0_2_4_6_8	Pin will function as an SCI.
IOPORT_PERIPHERAL_SCI1_3_5_7_9	Pin will function as an SCI.
IOPORT_PERIPHERAL_RSPI	Pin will function as a RSPI.
IOPORT_PERIPHERAL_RIIC	Pin will function as a RIIC.
IOPORT_PERIPHERAL_KEY	Pin will function as a KEY.
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC	Pin will function as a.
IOPORT_PERIPHERAL_CAC_AD	Pin will function as a CAC/ADC.
IOPORT_PERIPHERAL_BUS	Pin will function as a BUS.
IOPORT_PERIPHERAL_CTSU	Pin will function as a CTSU.
IOPORT_PERIPHERAL_LCDC	Pin will function as a segment LCD.

IOPORT_PERIPHERAL_DALI	Pin will function as a DALI.
IOPORT_PERIPHERAL_CAN	Pin will function as a CAN.
IOPORT_PERIPHERAL_QSPI	Pin will function as a QSPI.
IOPORT_PERIPHERAL_SSI	Pin will function as an SSI.
IOPORT_PERIPHERAL_USB_FS	Pin will function as a USB.
IOPORT_PERIPHERAL_USB_HS	Pin will function as a USB.
IOPORT_PERIPHERAL_SDHI_MMC	Pin will function as an SD/MMC.
IOPORT_PERIPHERAL_ETHER_MII	Pin will function as an Ethernet.
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet.
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC.
IOPORT_PERIPHERAL_LCD_GRAPHICS	Pin will function as a graphics.
IOPORT_PERIPHERAL_TRACE	Pin will function as a debug trace.
IOPORT_PERIPHERAL_END	Marks end of enum - used by.

◆ ioport_port_pin_t

enum ioport_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
IOPORT_PORT_00_PIN_00	IO port 0 pin 0.
IOPORT_PORT_00_PIN_01	IO port 0 pin 1.
IOPORT_PORT_00_PIN_02	IO port 0 pin 2.
IOPORT_PORT_00_PIN_03	IO port 0 pin 3.
IOPORT_PORT_00_PIN_04	IO port 0 pin 4.
IOPORT_PORT_00_PIN_05	IO port 0 pin 5.
IOPORT_PORT_00_PIN_06	IO port 0 pin 6.

IOPORT_PORT_00_PIN_07	IO port 0 pin 7.
IOPORT_PORT_00_PIN_08	IO port 0 pin 8.
IOPORT_PORT_00_PIN_09	IO port 0 pin 9.
IOPORT_PORT_00_PIN_10	IO port 0 pin 10.
IOPORT_PORT_00_PIN_11	IO port 0 pin 11.
IOPORT_PORT_00_PIN_12	IO port 0 pin 12.
IOPORT_PORT_00_PIN_13	IO port 0 pin 13.
IOPORT_PORT_00_PIN_14	IO port 0 pin 14.
IOPORT_PORT_00_PIN_15	IO port 0 pin 15.
IOPORT_PORT_01_PIN_00	IO port 1 pin 0.
IOPORT_PORT_01_PIN_01	IO port 1 pin 1.
IOPORT_PORT_01_PIN_02	IO port 1 pin 2.
IOPORT_PORT_01_PIN_03	IO port 1 pin 3.
IOPORT_PORT_01_PIN_04	IO port 1 pin 4.
IOPORT_PORT_01_PIN_05	IO port 1 pin 5.
IOPORT_PORT_01_PIN_06	IO port 1 pin 6.
IOPORT_PORT_01_PIN_07	IO port 1 pin 7.
IOPORT_PORT_01_PIN_08	IO port 1 pin 8.
IOPORT_PORT_01_PIN_09	IO port 1 pin 9.
IOPORT_PORT_01_PIN_10	IO port 1 pin 10.
IOPORT_PORT_01_PIN_11	IO port 1 pin 11.
IOPORT_PORT_01_PIN_12	IO port 1 pin 12.
IOPORT_PORT_01_PIN_13	IO port 1 pin 13.
IOPORT_PORT_01_PIN_14	IO port 1 pin 14.

IOPORT_PORT_01_PIN_15	IO port 1 pin 15.
IOPORT_PORT_02_PIN_00	IO port 2 pin 0.
IOPORT_PORT_02_PIN_01	IO port 2 pin 1.
IOPORT_PORT_02_PIN_02	IO port 2 pin 2.
IOPORT_PORT_02_PIN_03	IO port 2 pin 3.
IOPORT_PORT_02_PIN_04	IO port 2 pin 4.
IOPORT_PORT_02_PIN_05	IO port 2 pin 5.
IOPORT_PORT_02_PIN_06	IO port 2 pin 6.
IOPORT_PORT_02_PIN_07	IO port 2 pin 7.
IOPORT_PORT_02_PIN_08	IO port 2 pin 8.
IOPORT_PORT_02_PIN_09	IO port 2 pin 9.
IOPORT_PORT_02_PIN_10	IO port 2 pin 10.
IOPORT_PORT_02_PIN_11	IO port 2 pin 11.
IOPORT_PORT_02_PIN_12	IO port 2 pin 12.
IOPORT_PORT_02_PIN_13	IO port 2 pin 13.
IOPORT_PORT_02_PIN_14	IO port 2 pin 14.
IOPORT_PORT_02_PIN_15	IO port 2 pin 15.
IOPORT_PORT_03_PIN_00	IO port 3 pin 0.
IOPORT_PORT_03_PIN_01	IO port 3 pin 1.
IOPORT_PORT_03_PIN_02	IO port 3 pin 2.
IOPORT_PORT_03_PIN_03	IO port 3 pin 3.
IOPORT_PORT_03_PIN_04	IO port 3 pin 4.
IOPORT_PORT_03_PIN_05	IO port 3 pin 5.
IOPORT_PORT_03_PIN_06	IO port 3 pin 6.

IOPORT_PORT_03_PIN_07	IO port 3 pin 7.
IOPORT_PORT_03_PIN_08	IO port 3 pin 8.
IOPORT_PORT_03_PIN_09	IO port 3 pin 9.
IOPORT_PORT_03_PIN_10	IO port 3 pin 10.
IOPORT_PORT_03_PIN_11	IO port 3 pin 11.
IOPORT_PORT_03_PIN_12	IO port 3 pin 12.
IOPORT_PORT_03_PIN_13	IO port 3 pin 13.
IOPORT_PORT_03_PIN_14	IO port 3 pin 14.
IOPORT_PORT_03_PIN_15	IO port 3 pin 15.
IOPORT_PORT_04_PIN_00	IO port 4 pin 0.
IOPORT_PORT_04_PIN_01	IO port 4 pin 1.
IOPORT_PORT_04_PIN_02	IO port 4 pin 2.
IOPORT_PORT_04_PIN_03	IO port 4 pin 3.
IOPORT_PORT_04_PIN_04	IO port 4 pin 4.
IOPORT_PORT_04_PIN_05	IO port 4 pin 5.
IOPORT_PORT_04_PIN_06	IO port 4 pin 6.
IOPORT_PORT_04_PIN_07	IO port 4 pin 7.
IOPORT_PORT_04_PIN_08	IO port 4 pin 8.
IOPORT_PORT_04_PIN_09	IO port 4 pin 9.
IOPORT_PORT_04_PIN_10	IO port 4 pin 10.
IOPORT_PORT_04_PIN_11	IO port 4 pin 11.
IOPORT_PORT_04_PIN_12	IO port 4 pin 12.
IOPORT_PORT_04_PIN_13	IO port 4 pin 13.
IOPORT_PORT_04_PIN_14	IO port 4 pin 14.

IOPORT_PORT_04_PIN_15	IO port 4 pin 15.
IOPORT_PORT_05_PIN_00	IO port 5 pin 0.
IOPORT_PORT_05_PIN_01	IO port 5 pin 1.
IOPORT_PORT_05_PIN_02	IO port 5 pin 2.
IOPORT_PORT_05_PIN_03	IO port 5 pin 3.
IOPORT_PORT_05_PIN_04	IO port 5 pin 4.
IOPORT_PORT_05_PIN_05	IO port 5 pin 5.
IOPORT_PORT_05_PIN_06	IO port 5 pin 6.
IOPORT_PORT_05_PIN_07	IO port 5 pin 7.
IOPORT_PORT_05_PIN_08	IO port 5 pin 8.
IOPORT_PORT_05_PIN_09	IO port 5 pin 9.
IOPORT_PORT_05_PIN_10	IO port 5 pin 10.
IOPORT_PORT_05_PIN_11	IO port 5 pin 11.
IOPORT_PORT_05_PIN_12	IO port 5 pin 12.
IOPORT_PORT_05_PIN_13	IO port 5 pin 13.
IOPORT_PORT_05_PIN_14	IO port 5 pin 14.
IOPORT_PORT_05_PIN_15	IO port 5 pin 15.
IOPORT_PORT_06_PIN_00	IO port 6 pin 0.
IOPORT_PORT_06_PIN_01	IO port 6 pin 1.
IOPORT_PORT_06_PIN_02	IO port 6 pin 2.
IOPORT_PORT_06_PIN_03	IO port 6 pin 3.
IOPORT_PORT_06_PIN_04	IO port 6 pin 4.
IOPORT_PORT_06_PIN_05	IO port 6 pin 5.
IOPORT_PORT_06_PIN_06	IO port 6 pin 6.

IOPORT_PORT_06_PIN_07	IO port 6 pin 7.
IOPORT_PORT_06_PIN_08	IO port 6 pin 8.
IOPORT_PORT_06_PIN_09	IO port 6 pin 9.
IOPORT_PORT_06_PIN_10	IO port 6 pin 10.
IOPORT_PORT_06_PIN_11	IO port 6 pin 11.
IOPORT_PORT_06_PIN_12	IO port 6 pin 12.
IOPORT_PORT_06_PIN_13	IO port 6 pin 13.
IOPORT_PORT_06_PIN_14	IO port 6 pin 14.
IOPORT_PORT_06_PIN_15	IO port 6 pin 15.
IOPORT_PORT_07_PIN_00	IO port 7 pin 0.
IOPORT_PORT_07_PIN_01	IO port 7 pin 1.
IOPORT_PORT_07_PIN_02	IO port 7 pin 2.
IOPORT_PORT_07_PIN_03	IO port 7 pin 3.
IOPORT_PORT_07_PIN_04	IO port 7 pin 4.
IOPORT_PORT_07_PIN_05	IO port 7 pin 5.
IOPORT_PORT_07_PIN_06	IO port 7 pin 6.
IOPORT_PORT_07_PIN_07	IO port 7 pin 7.
IOPORT_PORT_07_PIN_08	IO port 7 pin 8.
IOPORT_PORT_07_PIN_09	IO port 7 pin 9.
IOPORT_PORT_07_PIN_10	IO port 7 pin 10.
IOPORT_PORT_07_PIN_11	IO port 7 pin 11.
IOPORT_PORT_07_PIN_12	IO port 7 pin 12.
IOPORT_PORT_07_PIN_13	IO port 7 pin 13.
IOPORT_PORT_07_PIN_14	IO port 7 pin 14.

IOPORT_PORT_07_PIN_15	IO port 7 pin 15.
IOPORT_PORT_08_PIN_00	IO port 8 pin 0.
IOPORT_PORT_08_PIN_01	IO port 8 pin 1.
IOPORT_PORT_08_PIN_02	IO port 8 pin 2.
IOPORT_PORT_08_PIN_03	IO port 8 pin 3.
IOPORT_PORT_08_PIN_04	IO port 8 pin 4.
IOPORT_PORT_08_PIN_05	IO port 8 pin 5.
IOPORT_PORT_08_PIN_06	IO port 8 pin 6.
IOPORT_PORT_08_PIN_07	IO port 8 pin 7.
IOPORT_PORT_08_PIN_08	IO port 8 pin 8.
IOPORT_PORT_08_PIN_09	IO port 8 pin 9.
IOPORT_PORT_08_PIN_10	IO port 8 pin 10.
IOPORT_PORT_08_PIN_11	IO port 8 pin 11.
IOPORT_PORT_08_PIN_12	IO port 8 pin 12.
IOPORT_PORT_08_PIN_13	IO port 8 pin 13.
IOPORT_PORT_08_PIN_14	IO port 8 pin 14.
IOPORT_PORT_08_PIN_15	IO port 8 pin 15.
IOPORT_PORT_09_PIN_00	IO port 9 pin 0.
IOPORT_PORT_09_PIN_01	IO port 9 pin 1.
IOPORT_PORT_09_PIN_02	IO port 9 pin 2.
IOPORT_PORT_09_PIN_03	IO port 9 pin 3.
IOPORT_PORT_09_PIN_04	IO port 9 pin 4.
IOPORT_PORT_09_PIN_05	IO port 9 pin 5.
IOPORT_PORT_09_PIN_06	IO port 9 pin 6.

IOPORT_PORT_09_PIN_07	IO port 9 pin 7.
IOPORT_PORT_09_PIN_08	IO port 9 pin 8.
IOPORT_PORT_09_PIN_09	IO port 9 pin 9.
IOPORT_PORT_09_PIN_10	IO port 9 pin 10.
IOPORT_PORT_09_PIN_11	IO port 9 pin 11.
IOPORT_PORT_09_PIN_12	IO port 9 pin 12.
IOPORT_PORT_09_PIN_13	IO port 9 pin 13.
IOPORT_PORT_09_PIN_14	IO port 9 pin 14.
IOPORT_PORT_09_PIN_15	IO port 9 pin 15.
IOPORT_PORT_10_PIN_00	IO port 10 pin 0.
IOPORT_PORT_10_PIN_01	IO port 10 pin 1.
IOPORT_PORT_10_PIN_02	IO port 10 pin 2.
IOPORT_PORT_10_PIN_03	IO port 10 pin 3.
IOPORT_PORT_10_PIN_04	IO port 10 pin 4.
IOPORT_PORT_10_PIN_05	IO port 10 pin 5.
IOPORT_PORT_10_PIN_06	IO port 10 pin 6.
IOPORT_PORT_10_PIN_07	IO port 10 pin 7.
IOPORT_PORT_10_PIN_08	IO port 10 pin 8.
IOPORT_PORT_10_PIN_09	IO port 10 pin 9.
IOPORT_PORT_10_PIN_10	IO port 10 pin 10.
IOPORT_PORT_10_PIN_11	IO port 10 pin 11.
IOPORT_PORT_10_PIN_12	IO port 10 pin 12.
IOPORT_PORT_10_PIN_13	IO port 10 pin 13.
IOPORT_PORT_10_PIN_14	IO port 10 pin 14.

IOPORT_PORT_10_PIN_15	IO port 10 pin 15.
IOPORT_PORT_11_PIN_00	IO port 11 pin 0.
IOPORT_PORT_11_PIN_01	IO port 11 pin 1.
IOPORT_PORT_11_PIN_02	IO port 11 pin 2.
IOPORT_PORT_11_PIN_03	IO port 11 pin 3.
IOPORT_PORT_11_PIN_04	IO port 11 pin 4.
IOPORT_PORT_11_PIN_05	IO port 11 pin 5.
IOPORT_PORT_11_PIN_06	IO port 11 pin 6.
IOPORT_PORT_11_PIN_07	IO port 11 pin 7.
IOPORT_PORT_11_PIN_08	IO port 11 pin 8.
IOPORT_PORT_11_PIN_09	IO port 11 pin 9.
IOPORT_PORT_11_PIN_10	IO port 11 pin 10.
IOPORT_PORT_11_PIN_11	IO port 11 pin 11.
IOPORT_PORT_11_PIN_12	IO port 11 pin 12.
IOPORT_PORT_11_PIN_13	IO port 11 pin 13.
IOPORT_PORT_11_PIN_14	IO port 11 pin 14.
IOPORT_PORT_11_PIN_15	IO port 11 pin 15.

◆ ioport_port_t

enum ioport_port_t	
Superset list of all possible IO ports.	
Enumerator	
IOPORT_PORT_00	IO port 0.
IOPORT_PORT_01	IO port 1.
IOPORT_PORT_02	IO port 2.
IOPORT_PORT_03	IO port 3.
IOPORT_PORT_04	IO port 4.
IOPORT_PORT_05	IO port 5.
IOPORT_PORT_06	IO port 6.
IOPORT_PORT_07	IO port 7.
IOPORT_PORT_08	IO port 8.
IOPORT_PORT_09	IO port 9.
IOPORT_PORT_10	IO port 10.
IOPORT_PORT_11	IO port 11.

ioport_pin_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I/O Port Interface

#include <r_ioport_api.h>

Data Fields

uint32_t	pin_cfg Pin PFS configuration - Use ioport_cfg_options_t parameters to configure.

ioport_port_pin_t	pin Pin identifier.

Detailed Description

Pin identifier and pin PFS pin configuration value

The documentation for this struct was generated from the following file:

- r_ioport_api.h

ioport_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I/O Port Interface

#include <r_ioport_api.h>

Data Fields

uint16_t	number_of_pins Number of pins for which there is configuration data.
ioport_pin_cfg_t const *	p_pin_cfg_data Pin configuration data.

Detailed Description

Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init()

The documentation for this struct was generated from the following file:

- r_ioport_api.h

ioport_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I/O Port Interface

#include <r_ioport_api.h>

Data Fields

ssp_err_t(*	init)(const ioport_cfg_t *p_cfg)
ssp_err_t(*	pinsCfg)(const ioport_cfg_t *p_cfg)
ssp_err_t(*	pinCfg)(ioport_port_pin_t pin, uint32_t cfg)
ssp_err_t(*	pinDirectionSet)(ioport_port_pin_t pin, ioport_direction_t direction)
ssp_err_t(*	pinEventInputRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_event)
ssp_err_t(*	pinEventOutputWrite)(ioport_port_pin_t pin, ioport_level_t pin_value)
ssp_err_t(*	pinEthernetModeCfg)(ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)
ssp_err_t(*	pinRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_value)
ssp_err_t(*	pinWrite)(ioport_port_pin_t pin, ioport_level_t level)
ssp_err_t(*	portDirectionSet)(ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)
ssp_err_t(*	portEventInputRead)(ioport_port_t port, ioport_size_t *p_event_data)
ssp_err_t(*	portEventOutputWrite)(ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
ssp_err_t(*	portRead)(ioport_port_t port, ioport_size_t *p_port_value)
ssp_err_t(*	portWrite)(ioport_port_t port, ioport_size_t value, ioport_size_t mask)
ssp_err_t(*	versionGet)(ssp_version_t *p_data)

Detailed Description

IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API.

Field Documentation

◆ init

ssp_err_t(* ioport_api_t::init) (const ioport_cfg_t *p_cfg)

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use ioport_api_t::pinsCfg for runtime reconfiguration of multiple pins.

Implemented as

- R_IOPORT_Init()

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ pinCfg

ssp_err_t(* ioport_api_t::pinCfg) (ioport_port_pin_t pin, uint32_t cfg)

Configure settings for an individual pin.

Implemented as

- R_IOPORT_PinCfg()

Parameters

[in]	pin	Pin to be read.
[in]	cfg	Configuration options for the pin.

◆ pinDirectionSet

ssp_err_t(* ioport_api_t::pinDirectionSet) (ioport_port_pin_t pin, ioport_direction_t direction)

Set the pin direction of a pin.

Implemented as

- R_IOPORT_PinDirectionSet()

Parameters

[in]	pin	Pin being configured.
[in]	direction	Direction to set pin to which is a member of ioport_direction_t.

◆ pinEthernetModeCfg

ssp_err_t(* ioport_api_t::pinEthernetModeCfg) (ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)

Configure the PHY mode of the Ethernet channels.

Implemented as

- R_IOPORT_EthernetModeCfg()

Parameters

[in]	channel	Channel configuration will be set for.
[in]	mode	PHY mode to set the channel to.

◆ pinEventInputRead

ssp_err_t(* ioport_api_t::pinEventInputRead) (ioport_port_pin_t pin, ioport_level_t *p_pin_event)

Read the event input data of the specified pin and return the level.

Implemented as

- R_IOPORT_PinEventInputRead()

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_event	Pointer to return the event data.

◆ pinEventOutputWrite

ssp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_port_pin_t pin, ioport_level_t pin_value)

Write pin event data.

Implemented as

- R_IOPORT_PinEventOutputWrite()

Parameters

[in]	pin	Pin event data is to be written to.
[in]	pin_value	Level to be written to pin output event.

◆ pinRead

ssp_err_t(* ioport_api_t::pinRead) (ioport_port_pin_t pin, ioport_level_t *p_pin_value)

Read level of a pin.

Implemented as

- R_IOPORT_PinRead()

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_value	Pointer to return the pin level.

◆ pinsCfg

ssp_err_t(* ioport_api_t::pinsCfg) (const ioport_cfg_t *p_cfg)

Configure multiple pins.

Implemented as

- R_IOPORT_PinsCfg()

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ pinWrite

ssp_err_t(* ioport_api_t::pinWrite) (ioport_port_pin_t pin, ioport_level_t level)

Write specified level to a pin.

Implemented as

- R_IOPORT_PinWrite()

Parameters

[in]	pin	Pin to be written to.
[in]	level	State to be written to the pin.

◆ portDirectionSet

ssp_err_t(* ioport_api_t::portDirectionSet) (ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)

Set the direction of one or more pins on a port.

Implemented as

- R_IOPORT_PortDirectionSet()

Parameters

[in]	port	Port being configured.
[in]	direction_values	Value controlling direction of pins on port (1 - output, 0 - input).
[in]	mask	Mask controlling which pins on the port are to be configured.

◆ portEventInputRead

ssp_err_t(* ioport_api_t::portEventInputRead) (ioport_port_t port, ioport_size_t *p_event_data)

Read captured event data for a port.

Implemented as

- R_IOPORT_PortEventInputRead()

Parameters

[in]	port	Port to be read.
[in]	p_event_data	Pointer to return the event data.

◆ portEventOutputWrite

ssp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

Write event output data for a port.

Implemented as

- R_IOPORT_PortEventOutputWrite()

Parameters

[in]	port	Port event data will be written to.
[in]	event_data	Data to be written as event data to specified port.
[in]	mask_value	Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port.

◆ portRead

ssp_err_t(* ioport_api_t::portRead) (ioport_port_t port, ioport_size_t *p_port_value)

Read states of pins on the specified port.

Implemented as

- R_IOPORT_PortRead()

Parameters

[in]	port	Port to be read.
[in]	p_port_value	Pointer to return the port value.

◆ portWrite

ssp_err_t(* ioport_api_t::portWrite) (ioport_port_t port, ioport_size_t value, ioport_size_t mask)

Write to multiple pins on a port.

Implemented as

- R_IOPORT_PortWrite()

Parameters

[in]	port	Port to be written to.
[in]	value	Value to be written to the port.
[in]	mask	Mask controlling which pins on the port are written to.

◆ versionGet

ssp_err_t(* ioport_api_t::versionGet) (ssp_version_t *p_data)

Return the version of the IOPort driver.

Implemented as

- R_IOPORT_VersionGet()

Parameters

[out]	p_data	Memory address to return version information to.

The documentation for this struct was generated from the following file:

- r_ioport_api.h

ioport_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » I/O Port Interface

#include <r_ioport_api.h>

Data Fields

ioport_cfg_t const * p_cfg

	Pointer to the configuration structure for this instance.
ioport_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_ioport_api.h

5.1.4.21 JPEG Decode Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for JPEG decode functions. More...

Data Structures

struct	jpeg_decode_callback_args_t
struct	jpeg_decode_cfg_t
struct	jpeg_decode_api_t
struct	jpeg_decode_instance_t

Macros

#define	JPEG_DECODE_API_VERSION_MAJOR (2U)

Typedefs

typedef void	jpeg_decode_ctrl_t

Enumerations

enum	jpeg_decode_color_space_t { JPEG_DECODE_COLOR_SPACE_YCBCR444 = 0, JPEG_DECODE_COLOR_SPACE_YCBCR422 = 1, JPEG_DECODE_COLOR_SPACE_YCBCR420 = 2, JPEG_DECODE_COLOR_SPACE_YCBCR411 = 6 }

enum	jpeg_decode_data_format_t { JPEG_DECODE_DATA_FORMAT_NORMAL = 0, JPEG_DECODE_DATA_FORMAT_BYTE_SWAP, JPEG_DECODE_DATA_FORMAT_WORD_SWAP, JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP, JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP, JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP, JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP, JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP }
enum	jpeg_decode_pixel_format_t { JPEG_DECODE_PIXEL_FORMAT_ARGB8888 = 1, JPEG_DECODE_PIXEL_FORMAT_RGB565 }
enum	jpeg_decode_status_t { JPEG_DECODE_STATUS_FREE = 0x0, JPEG_DECODE_STATUS_IDLE = 0x1, JPEG_DECODE_STATUS_RUNNING = 0x2, JPEG_DECODE_STATUS_DONE = 0x4, JPEG_DECODE_STATUS_INPUT_PAUSE = 0x8, JPEG_DECODE_STATUS_OUTPUT_PAUSE = 0x10, JPEG_DECODE_STATUS_IMAGE_SIZE_READY = 0x20, JPEG_DECODE_STATUS_ERROR = 0x40, JPEG_DECODE_STATUS_HEADER_PROCESSING = 0x80 }
enum	jpeg_decode_subsample_t { JPEG_DECODE_OUTPUT_NO_SUBSAMPLE = 0, JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF, JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER, JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH }
enum	jpeg_decode_count_enable_t { JPEG_DECODE_COUNT_DISABLE = 0, JPEG_DECODE_COUNT_ENABLE }
enum	jpeg_decode_resume_mode_t { JPEG_DECODE_COUNT_MODE_ADDRESS_CONTINUE = 0, JPEG_DECODE_COUNT_MODE_ADDRESS_REINITIALIZE }

Detailed Description

Interface for JPEG decode functions.

Summary

The JPEG DECODE interface provides JPEG decoder functionality. It allows application to convert a JPEG image into bitmap data suitable for display frame buffer.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers

- Using SSP Modules

JPEG DECODE Interface description: JPEG Decode Driver

Macro Definition Documentation

◆ JPEG_DECODE_API_VERSION_MAJOR

#define JPEG_DECODE_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes. Configuration for this module

Typedef Documentation

◆ jpeg_decode_ctrl_t

typedef void jpeg_decode_ctrl_t
JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. **Implemented as** ◦ jpeg_decode_instance_ctrl_t

Enumeration Type Documentation

◆ jpeg_decode_color_space_t

enum jpeg_decode_color_space_t	
Image color space definitions	
Enumerator	
JPEG_DECODE_COLOR_SPACE_YCBCR444	Color Space YCbCr 444.
JPEG_DECODE_COLOR_SPACE_YCBCR422	Color Space YCbCr 422.
JPEG_DECODE_COLOR_SPACE_YCBCR420	Color Space YCbCr 420.
JPEG_DECODE_COLOR_SPACE_YCBCR411	Color Space YCbCr 411.

◆ jpeg_decode_count_enable_t

enum jpeg_decode_count_enable_t	
Data type for decoding count mode enable.	
Enumerator	
JPEG_DECODE_COUNT_DISABLE	Count mode disable.
JPEG_DECODE_COUNT_ENABLE	Count mode enable.

◆ jpeg_decode_data_format_t

enum jpeg_decode_data_format_t	
Multi-byte Data Format	
Enumerator	
JPEG_DECODE_DATA_FORMAT_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_DECODE_DATA_FORMAT_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_DECODE_DATA_FORMAT_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap

◆ jpeg_decode_pixel_format_t

enum jpeg_decode_pixel_format_t	
Pixel Data Format	
Enumerator	
JPEG_DECODE_PIXEL_FORMAT_ARGB8888	Pixel Data ARGB8888 format.
JPEG_DECODE_PIXEL_FORMAT_RGB565	Pixel Data RGB565 format.

◆ jpeg_decode_resume_mode_t

enum jpeg_decode_resume_mode_t	
Data type for decoding count mode enable.	
Enumerator	
JPEG_DECODE_COUNT_MODE_ADDRESS_CONTINUE	The data buffer address will not be initialized when resuming image data lines.
JPEG_DECODE_COUNT_MODE_ADDRESS_REINITIALIZE	The data buffer address will be initialized when resuming image data lines.

◆ jpeg_decode_status_t

enum jpeg_decode_status_t	
JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status	
Enumerator	
JPEG_DECODE_STATUS_FREE	JPEG codec module is not yet open.
JPEG_DECODE_STATUS_IDLE	JPEG Codec module is open, and is not operational.
JPEG_DECODE_STATUS_RUNNING	JPEG Codec is running.
JPEG_DECODE_STATUS_DONE	JPEG Codec has successfully finished the operation.
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG Codec paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG decoding operation obtained image size, and paused.
JPEG_DECODE_STATUS_ERROR	JPEG Codec module encountered an error.
JPEG_DECODE_STATUS_HEADER_PROCESSING	JPEG Codec module is reading the JPEG header information.

◆ jpeg_decode_subsample_t

enum jpeg_decode_subsample_t	
Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.	
Enumerator	
JPEG_DECODE_OUTPUT_NO_SUBSAMPLE	No subsample. The image is decoded with no reduction in size.
JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF	The output image size is reduced by half.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER	The output image size is reduced to one-quarter.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH	The output image size is reduced to one-eighth.

jpeg_decode_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Decode Interface

#include <r_jpeg_decode_api.h>

Data Fields

jpeg_decode_status_t	status JPEG status.
void const *	p_context Pointer to user-provided context.

Detailed Description

Callback status structure

The documentation for this struct was generated from the following file:

- r_jpeg_decode_api.h

jpeg_decode_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Decode Interface

#include <r_jpeg_decode_api.h>

Data Fields

jpeg_decode_color_space_t	color_space Color space.
jpeg_decode_data_format_t	input_data_format Input data stream byte order.
jpeg_decode_data_format_t	output_data_format Output data stream byte order.
jpeg_decode_pixel_format_t	pixel_format Pixel format.
uint8_t	alpha_value Alpha value to be applied to decoded pixel data. Only valid for ARGB888 format.
uint8_t	jdti_ipl Data transfer interrupt priority.
uint8_t	jedi_ipl Decompression interrupt priority.
void(*	p_callback)(jpeg_decode_callback_args_t *p_args) User-supplied callback functions.
void const *	p_context Placeholder for user data. Passed to user callback in

	jpeg_decode_callback_args_t.

Detailed Description

User configuration structure, used in open function.

The documentation for this struct was generated from the following file:

- r_jpeg_decode_api.h

jpeg_decode_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Decode Interface

#include <r_jpeg_decode_api.h>

Data Fields

ssp_err_t(*	open)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
ssp_err_t(*	outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
ssp_err_t(*	horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
ssp_err_t(*	imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
ssp_err_t(*	inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
ssp_err_t(*	linesDecodedGet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
ssp_err_t(*	imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
ssp_err_t(*	statusGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
ssp_err_t(*	close)(jpeg_decode_ctrl_t *const p_ctrl)

ssp_err_t(*	versionGet)(ssp_version_t *p_version)
ssp_err_t(*	pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

Detailed Description

JPEG functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* jpeg_decode_api_t::close) (jpeg_decode_ctrl_t *const p_ctrl)

Cancel an outstanding operation.

Implemented as

- R_JPEG_Decode_Close()

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::Open call.

◆ horizontalStrideSet

ssp_err_t(* jpeg_decode_api_t::horizontalStrideSet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

Configure the horizontal stride value.

Implemented as

- R_JPEG_Decode_HorizontalStrideSet()

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	horizontal_stride	Horizontal stride value to be used for the decoded image data.
[in]	buffer_size	Size of the output buffer

◆ imageSizeGet

ssp_err_t(* jpeg_decode_api_t::imageSizeGet) (jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Retrieve image size during decoding operation.

Implemented as

- R_JPEG_Decode_ImageSizeGet()

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_horizontal_size	Image horizontal size, in number of pixels.
[out]	p_vertical_size	Image vertical size, in number of pixels.

◆ imageSubsampleSet

ssp_err_t(* jpeg_decode_api_t::imageSubsampleSet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Configure the horizontal and vertical subsample settings.

Implemented as

- R_JPEG_Decode_ImageSubsampleSet()

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	horizontal_subsample	Horizontal subsample value
[in]	vertical_subsample	Vertical subsample value

◆ inputBufferSet

ssp_err_t(* jpeg_decode_api_t::inputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign input data buffer to JPEG codec.

Implemented as

- R_JPEG_Decode_InputBufferSet()

Precondition

the JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	p_buffer	Pointer to the input buffer space
[in]	buffer_size	Size of the input buffer

◆ linesDecodedGet

ssp_err_t(* jpeg_decode_api_t::linesDecodedGet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)

Return the number of lines decoded into the output buffer.

Implemented as

- R_JPEG_Decode_LinesDecodedGet()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_lines	Number of lines decoded

◆ open

ssp_err_t(* jpeg_decode_api_t::open) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)

Initial configuration

Implemented as

- R_JPEG_Decode_Open()

Precondition

none

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ outputBufferSet

ssp_err_t(* jpeg_decode_api_t::outputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign output buffer to JPEG codec for storing output data.

Implemented as

- R_JPEG_Decode_OutputBufferSet()

Precondition

The JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned. For the decoding process, the HLD driver automatically computes the number of lines of the image to decoded so the output data fits into the given space. If the supplied output buffer is not able to hold the entire frame, the application should call the Output Full Callback function so it can be notified when additional buffer space is needed.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	p_buffer	Pointer to the output buffer space
[in]	buffer_size	Size of the output buffer

◆ pixelFormatGet

ssp_err_t(* jpeg_decode_api_t::pixelFormatGet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

Get the input pixel format.

Implemented as

- R_JPEG_Decode_PixelFormatGet()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_color_space	JPEG input format.

◆ statusGet

ssp_err_t(* jpeg_decode_api_t::statusGet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)

Retrieve current status of the JPEG codec module.

Implemented as

- R_JPEG_Decode_StatusGet()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_status	JPEG module status

◆ versionGet

ssp_err_t(* jpeg_decode_api_t::versionGet) (ssp_version_t *p_version)

Get version and store it in provided pointer p_version.

Implemented as

- R_JPEG_Decode_VersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_jpeg_decode_api.h

jpeg_decode_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Decode Interface

#include <r_jpeg_decode_api.h>

Data Fields

jpeg_decode_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
jpeg_decode_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
jpeg_decode_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_jpeg_decode_api.h

5.1.4.22 JPEG Encode Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for JPEG encode functions. More...

Data Structures

struct	jpeg_encode_raw_image_parameters
struct	jpeg_encode_callback_args_t
struct	jpeg_encode_cfg_t
struct	jpeg_encode_api_t
struct	jpeg_encode_instance_t

Macros

#define	JPEG_ENCODE_API_VERSION_MAJOR (2U)

Typedefs

typedef void	jpeg_encode_ctrl_t

Enumerations

enum jpeg_encode_data_format_t {
JPEG_ENCODE_DATA_FORMAT_NORMAL = 0,
JPEG_ENCODE_DATA_FORMAT_BYTE_SWAP,
JPEG_ENCODE_DATA_FORMAT_WORD_SWAP,
JPEG_ENCODE_DATA_FORMAT_WORD_BYTE_SWAP,
JPEG_ENCODE_DATA_FORMAT_LONGWORD_SWAP,
JPEG_ENCODE_DATA_FORMAT_LONGWORD_BYTE_SWAP,
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_SWAP,
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP,
JPEG_ENCODE_DATA_FORMAT_MAX
}

enum jpeg_encode_status_t {
JPEG_ENCODE_STATUS_FREE = 0x0, JPEG_ENCODE_STATUS_IDLE = 0x1, JPEG_ENCODE_STATUS_RUNNING = 0x2,
JPEG_ENCODE_STATUS_DONE = 0x4,
JPEG_ENCODE_STATUS_INPUT_PAUSE = 0x8
}

enum jpeg_encode_count_t { JPEG_ENCODE_COUNT_DISABLE = 0, JPEG_ENCODE_COUNT_ENABLE }

enum jpeg_encode_resume_mode_t {
JPEG_ENCODE_COUNT_MODE_ADDRESS_CONTINUE = 0,
JPEG_ENCODE_COUNT_MODE_ADDRESS_REINITIALIZE }

Detailed Description

Interface for JPEG encode functions.

Summary

The JPEG ENCODE interface provides JPEG encoder functionality. It allows application to convert a JPEG image into bitmap data suitable for display frame buffer.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

JPEG ENCODE Interface description: JPEG Encode Driver

Macro Definition Documentation

◆ JPEG_ENCODE_API_VERSION_MAJOR

#define JPEG_ENCODE_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ jpeg_encode_ctrl_t

typedef void jpeg_encode_ctrl_t
JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. **Implemented as** ◦ jpeg_encode_instance_ctrl_t

Enumeration Type Documentation

◆ jpeg_encode_count_t

enum jpeg_encode_count_t	
Data type for encoding count mode enable.	
Enumerator	
JPEG_ENCODE_COUNT_DISABLE	Count mode disable.
JPEG_ENCODE_COUNT_ENABLE	Count mode enable.

◆ jpeg_encode_data_format_t

enum jpeg_encode_data_format_t	
Multi-byte Data Format	
Enumerator	
JPEG_ENCODE_DATA_FORMAT_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_ENCODE_DATA_FORMAT_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_ENCODE_DATA_FORMAT_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_ENCODE_DATA_FORMAT_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap
JPEG_ENCODE_DATA_FORMAT_MAX	Maximum value of data format.

◆ jpeg_encode_resume_mode_t

enum jpeg_encode_resume_mode_t	
Data type for encoding resume mode	
Enumerator	
JPEG_ENCODE_COUNT_MODE_ADDRESS_CONTINUE	The data buffer address will not be initialized when resuming image data lines.
JPEG_ENCODE_COUNT_MODE_ADDRESS_REINITIALIZE	The data buffer address will be initialized when resuming image data lines.

◆ jpeg_encode_status_t

enum jpeg_encode_status_t	
JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status	
Enumerator	
JPEG_ENCODE_STATUS_FREE	JPEG codec module is not yet open.
JPEG_ENCODE_STATUS_IDLE	JPEG Codec module is open, and is not operational.
JPEG_ENCODE_STATUS_RUNNING	JPEG Codec is running.
JPEG_ENCODE_STATUS_DONE	JPEG Codec has successfully finished the operation.
JPEG_ENCODE_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.

jpeg_encode_raw_image_parameters Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Encode Interface

#include <r_jpeg_encode_api.h>

Data Fields

uint16_t	horizontal_stride Horizontal stride.
uint16_t	horizontal_resolution Horizontal Resolution in pixel.
uint16_t	vertical_resolution Vertical Resolution in pixel.

Detailed Description

Image parameter structure

The documentation for this struct was generated from the following file:

- r_jpeg_encode_api.h

jpeg_encode_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Encode Interface

#include <r_jpeg_encode_api.h>

Data Fields

volatile jpeg_encode_status_t	status JPEG status.
uint32_t	image_size JPEG image size.
void const *	p_context Pointer to user-provided context.

Detailed Description

Callback status structure

The documentation for this struct was generated from the following file:

- r_jpeg_encode_api.h

jpeg_encode_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Encode Interface

#include <r_jpeg_encode_api.h>

Data Fields

jpeg_encode_data_format_t	input_data_format Input data stream byte order.
jpeg_encode_data_format_t	output_data_format Output data stream byte order.
uint16_t	dri_marker DRI Marker setting 0 :- No DRI and RST marker.
uint8_t	jdti_ipl Data transfer interrupt priority.
uint8_t	jedi_ipl Decompression interrupt priority.
uint8_t	quality_factor JPEG image quality.
uint16_t	vertical_resolution vertical resolution of input image
uint16_t	horizontal_resolution horizontal resolution of input image
uint8_t const *	p_quant_luma_table Luma table.
uint8_t const *	p_quant_croma_table croma table
uint8_t const *	p_huffman_luma_ac_table Huffman AC table for luma.

uint8_t const *	p_huffman_luma_dc_table Huffman DC table for luma.
uint8_t const *	p_huffman_croma_ac_table Huffman AC table for croma.
uint8_t const *	p_huffman_croma_dc_table Huffman DC table for croma.
void(*	p_callback)(jpeg_encode_callback_args_t *p_args) User-supplied callback functions.
void const *	p_context Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.

Detailed Description

User configuration structure, used in open function.

The documentation for this struct was generated from the following file:

- r_jpeg_encode_api.h

jpeg_encode_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Encode Interface

#include <r_jpeg_encode_api.h>

Data Fields

ssp_err_t(*	open)(jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_cfg_t const *const p_cfg)
ssp_err_t(*	imageParameterSet)(jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_raw_image_parameters *p_raw_image_parameters)

ssp_err_t(*	outputBufferSet)(jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer)
ssp_err_t(*	inputBufferSet)(jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
ssp_err_t(*	statusGet)(jpeg_encode_ctrl_t *const p_ctrl, volatile jpeg_encode_status_t *p_status)
ssp_err_t(*	close)(jpeg_encode_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

JPEG functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* jpeg_encode_api_t::close) (jpeg_encode_ctrl_t *const p_ctrl)

Cancel an outstanding operation.

Implemented as

- R_JPEG_Encode_Close()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_encode_api_t::Open call.

◆ imageParameterSet

ssp_err_t(* jpeg_encode_api_t::imageParameterSet) (jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_raw_image_parameters *p_raw_image_parameters)

Set image parameters to JPEG Codec

Implemented as

- R_JPEG_Encode_ImageParameterSet()

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_raw_image_parameters	Pointer to the RAW image parameters

◆ inputBufferSet

ssp_err_t(* jpeg_encode_api_t::inputBufferSet) (jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign input data buffer to JPEG codec.

Implemented as

- R_JPEG_Encode_InputBufferSet()

Precondition

the JPEG codec module must have been opened properly, output buffer and image parameter must be set prior to call this function.

Note

The buffer starting address must be 8-byte aligned.

Parameters

[in]	p_ctrl	Control block set in jpeg_encode_api_t::open call.
[in]	p_buffer	Pointer to the input buffer space
[in]	buffer_size	Size of the input buffer

◆ open

ssp_err_t(* jpeg_encode_api_t::open) (jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_cfg_t const *const p_cfg)

Initial configuration

Implemented as

- R_JPEG_Encode_Open()

Precondition

none

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ outputBufferSet

ssp_err_t(* jpeg_encode_api_t::outputBufferSet) (jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer)

Assign output buffer to JPEG codec for storing output data.

Implemented as

- R_JPEG_Encode_OutputBufferSet()

Precondition

The JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned.

Parameters

[in]	p_ctrl	Control block set in jpeg_encode_api_t::open call.
[in]	p_buffer	Pointer to the output buffer space

◆ statusGet

ssp_err_t(* jpeg_encode_api_t::statusGet) (jpeg_encode_ctrl_t *const p_ctrl, volatile jpeg_encode_status_t *p_status)

Retrieve current status of the JPEG codec module.

Implemented as

- R_JPEG_Encode_StatusGet()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_encode_api_t::open call.
[out]	p_status	JPEG module status

◆ versionGet

ssp_err_t(* jpeg_encode_api_t::versionGet) (ssp_version_t *p_version)

Get version and store it in provided pointer p_version.

Implemented as

- R_JPEG_Encode_VersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_jpeg_encode_api.h

jpeg_encode_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » JPEG Encode Interface

#include <r_jpeg_encode_api.h>

Data Fields

jpeg_encode_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
jpeg_encode_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
jpeg_encode_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_jpeg_encode_api.h

5.1.4.23 Key Matrix Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for key matrix functions. More...

Data Structures

struct	keymatrix_callback_args_t
struct	keymatrix_cfg_t
struct	keymatrix_api_t
struct	keymatrix_instance_t

Macros

#define	KEYMATRIX_API_VERSION_MAJOR (2U) KEY MATRIX API version number (Major)
#define	KEYMATRIX_API_VERSION_MINOR (0U)

	KEY MATRIX API version number (Minor)

Typedefs

typedef uint32_t	keymatrix_channels_t
typedef void	keymatrix_ctrl_t

Enumerations

enum	keymatrix_trigger_t { KEYMATRIX_TRIG_FALLING = 0, KEYMATRIX_TRIG_RISING = 1 }

Detailed Description

Interface for key matrix functions.

Summary

The KEYMATRIX interface provides standard KeyMatrix functionality including event generation on a rising or falling edge for one or more channels at the same time. The generated event indicates all channels that are active in that instant via a bit mask. This allows the interface to be used with a matrix configuration or a one-to-one hardware implementation that is triggered on either a rising or a falling edge.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Key Matrix Interface description: Key Matrix Driver

Typedef Documentation

◆ keymatrix_channels_t

typedef uint32_t keymatrix_channels_t
Channel definition. This is a bit mask with each bit from 0-7 representing channels 0-7 respectively.

◆ keymatrix_ctrl_t

typedef void keymatrix_ctrl_t
Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. **Implemented as** ◦ kint_instance_ctrl_t

Enumeration Type Documentation

◆ keymatrix_trigger_t

enum keymatrix_trigger_t	
Trigger type: rising edge, falling edge	
Enumerator	
KEYMATRIX_TRIG_FALLING	Falling edge trigger.
KEYMATRIX_TRIG_RISING	Rising edge trigger.

keymatrix_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Key Matrix Interface

#include <r_keymatrix_api.h>

Data Fields

void const *	p_context Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t.
keymatrix_channels_t	channels

Detailed Description

Callback function parameter data

Field Documentation

◆ channels

keymatrix_channels_t keymatrix_callback_args_t::channels

Bit vector representing the physical hardware channel(s) that caused the interrupt. The bit vector is used for compatibility with matrix designs where more than one input will be active at once.

Note

Not all HAL drivers support matrix mode. See r_kint.h for details.

The documentation for this struct was generated from the following file:

- r_keymatrix_api.h

keymatrix_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Key Matrix Interface

#include <r_keymatrix_api.h>

Data Fields

keymatrix_channels_t	channels Key Input channel(s). Bit mask of channels to open.
keymatrix_trigger_t	trigger Key Input trigger setting.
bool	autostart Start operation and enable interrupts during open().
void(*	p_callback)(keymatrix_callback_args_t *p_args) Callback for key interrupt ISR.
void const *	p_context Holder for user data. Passed to callback in keymatrix_user_cb_data_t.

void const *	p_extend Extension parameter for hardware specific settings.
uint8_t	irq_ipl Interrupt priority level.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- r_keymatrix_api.h

keymatrix_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Key Matrix Interface

#include <r_keymatrix_api.h>

Data Fields

ssp_err_t(*	open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
ssp_err_t(*	enable)(keymatrix_ctrl_t *const p_ctrl)
ssp_err_t(*	disable)(keymatrix_ctrl_t *const p_ctrl)
ssp_err_t(*	triggerSet)(keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)
ssp_err_t(*	close)(keymatrix_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API.

Field Documentation

◆ close

ssp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

- R_KINT_KEYMATRIX_Close()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ disable

ssp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)

Disable Key interrupt.

Implemented as

- R_KINT_KEYMATRIX_Disable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ enable

ssp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)

Enable Key interrupt

Implemented as

- R_KINT_KEYMATRIX_Enable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ open

ssp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_KINT_KEYMATRIX_Open()

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ triggerSet

ssp_err_t(* keymatrix_api_t::triggerSet) (keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)

Set trigger for Key interrupt.

Implemented as

- R_KINT_KEYMATRIX_TriggerSet()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.
[in]	trigger	Trigger source for key interrupt; defined in enumeration of keymatrix_trigger_t.

◆ versionGet

ssp_err_t(* keymatrix_api_t::versionGet) (ssp_version_t *const p_version)
Get version and store it in provided pointer p_version. **Implemented as** ◦ R_KINT_VersionGet() **Parameters** [out] p_version Code and API version used.

The documentation for this struct was generated from the following file:

- r_keymatrix_api.h

keymatrix_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Key Matrix Interface

#include <r_keymatrix_api.h>

Data Fields

keymatrix_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
keymatrix_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
keymatrix_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_keymatrix_api.h

5.1.4.24 Low Power Modes V2 Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for accessing low power modes. More...

Data Structures

struct	lpmv2_cfg_t
struct	lpmv2_api_t
struct	lpmv2_instance_t

Macros

#define	LPMV2_API_VERSION_MAJOR (3U)

Enumerations

enum	lpmv2_low_power_mode_t { LPMV2_LOW_POWER_MODE_SLEEP, LPMV2_LOW_POWER_MODE_STANDBY, LPMV2_LOW_POWER_MODE_STANDBY_SNOOZE, LPMV2_LOW_POWER_MODE_DEEP }

Detailed Description

Interface for accessing low power modes.

Summary

This section defines the API for the LPMV2 (Low Power Mode) Driver. The LPMV2 Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPMV2 driver supports configuration of MCU low power modes using the LPMV2 hardware peripheral. The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCUs.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

LPMV2 Interface description: HAL LPMV2 Interface

Macro Definition Documentation

◆ LPMV2_API_VERSION_MAJOR

#define LPMV2_API_VERSION_MAJOR (3U)
Register definitions, common services and error codes.

Enumeration Type Documentation

◆ lpmv2_low_power_mode_t

enum lpmv2_low_power_mode_t	
Low power modes	
Enumerator	
LPMV2_LOW_POWER_MODE_SLEEP	Sleep mode.
LPMV2_LOW_POWER_MODE_STANDBY	Software Standby mode.
LPMV2_LOW_POWER_MODE_STANDBY_SNOOZE	Software Standby mode with Snooze mode enabled.
LPMV2_LOW_POWER_MODE_DEEP	Deep Software Standby mode.

lpmv2_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Power Modes V2 Interface

#include <r_lpmv2_api.h>

Data Fields

lpmv2_low_power_mode_t	low_power_mode
void const *	p_extend

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ low_power_mode

lpmv2_low_power_mode_t lpmv2_cfg_t::low_power_mode
Low Power Mode

◆ p_extend

void const* lpmv2_cfg_t::p_extend
MCU Specific configuration

The documentation for this struct was generated from the following file:

- r_lpmv2_api.h

lpmv2_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Power Modes V2 Interface

#include <r_lpmv2_api.h>

Data Fields

ssp_err_t(*	init)(void)
ssp_err_t(*	lowPowerCfg)(lpmv2_cfg_t const *const p_cfg)
ssp_err_t(*	lowPowerModeEnter)(void)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	clearIOKeep)(void)

Detailed Description

lpmv2 driver structure. General lpmv2 functions implemented at the HAL layer will follow this API.

Field Documentation

◆ clearIOKeep

ssp_err_t(* lpmv2_api_t::clearIOKeep) (void)

Clear the IOKEEP bit after deep software standby.

- **Implemented as**

- R_LPMV2_ClearIOKeep()

◆ init

ssp_err_t(* lpmv2_api_t::init) (void)

Initialization function

Implemented as

- R_LPMV2_Init()

◆ lowPowerCfg

ssp_err_t(* lpmv2_api_t::lowPowerCfg) (lpmv2_cfg_t const *const p_cfg)

Configure a low power mode.

Implemented as

- R_LPMV2_LowPowerConfigure()

Parameters

[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ lowPowerModeEnter

ssp_err_t(* lpmv2_api_t::lowPowerModeEnter) (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

Implemented as

- R_LPMV2_LowPowerModeEnter()

◆ versionGet

ssp_err_t(* lpmv2_api_t::versionGet) (ssp_version_t *const p_version)		
Get the driver version based on compile time macros. **Implemented as** ◦ R_LPMV2_VersionGet() **Parameters**		
[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_lpmv2_api.h

lpmv2_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Power Modes V2 Interface

#include <r_lpmv2_api.h>

Data Fields

lpmv2_cfg_t const *const	p_cfg Pointer to the configuration structure for this instance.
lpmv2_api_t const *const	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_lpmv2_api.h

5.1.4.25 Low Voltage Detection Interface

Renesas Synergy Software Package Reference » HAL Interfaces

This section defines the API for the LVD (Low Voltage Detection) Driver. More...

Data Structures

struct	lvd_status_t
struct	lvd_callback_args_t
struct	lvd_cfg_t
struct	lvd_api_t
struct	lvd_instance_t

Macros

#define	LVD_API_VERSION_MAJOR (2U)

Typedefs

typedef void	lvd_ctrl_t

Enumerations

```
enum lvd_threshold_t {
  LVD_THRESHOLD_MONITOR_1_LEVEL_0 = 0x0UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_1 = 0x1UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_2 = 0x2UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_3 = 0x3UL,
  LVD_THRESHOLD_MONITOR_1_LEVEL_4 = 0x4UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_5 = 0x5UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_6 = 0x6UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_7 = 0x7UL,
  LVD_THRESHOLD_MONITOR_1_LEVEL_8 = 0x8UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_9 = 0x9UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_A = 0xAUL,
LVD_THRESHOLD_MONITOR_1_LEVEL_B = 0xBUL,
  LVD_THRESHOLD_MONITOR_1_LEVEL_C = 0xCUL,
LVD_THRESHOLD_MONITOR_1_LEVEL_D = 0xDUL,
LVD_THRESHOLD_MONITOR_1_LEVEL_E = 0xEUL,
LVD_THRESHOLD_MONITOR_1_LEVEL_F = 0xFUL,
  LVD_THRESHOLD_MONITOR_1_LEVEL_11 = 0x11UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_12 = 0x12UL,
LVD_THRESHOLD_MONITOR_1_LEVEL_13 = 0x13UL,
LVD_THRESHOLD_MONITOR_2_LEVEL_0 = 0x0UL,
  LVD_THRESHOLD_MONITOR_2_LEVEL_1 = 0x1UL,
LVD_THRESHOLD_MONITOR_2_LEVEL_2 = 0x2UL,
```

	LVD_THRESHOLD_MONITOR_2_LEVEL_3 = 0x3UL, LVD_THRESHOLD_MONITOR_2_LEVEL_5 = 0x5UL, LVD_THRESHOLD_MONITOR_2_LEVEL_6 = 0x6UL, LVD_THRESHOLD_MONITOR_2_LEVEL_7 = 0x7UL }
enum	lvd_response_t { LVD_RESPONSE_NMI, LVD_RESPONSE_INTERRUPT, LVD_RESPONSE_RESET, LVD_RESPONSE_NONE }
enum	lvd_voltage_slope_t { LVD_VOLTAGE_SLOPE_RISING = 0, LVD_VOLTAGE_SLOPE_FALLING = 1, LVD_VOLTAGE_SLOPE_BOTH = 2 }
enum	lvd_threshold_crossing_t { LVD_THRESHOLD_CROSSING_NOT_DETECTED = 0, LVD_THRESHOLD_CROSSING_DETECTED = 1 }
enum	lvd_current_state_t { LVD_CURRENT_STATE_BELOW_THRESHOLD = 0, LVD_CURRENT_STATE_ABOVE_THRESHOLD = 1 }

Detailed Description

This section defines the API for the LVD (Low Voltage Detection) Driver.

The LVD driver provides functions for configuring the LVD hardware peripheral.

The process of configuring and enabling a Low Voltage Detection monitor has very specific timing constraints and register write ordering. Because of these constraints, the entire process of configuring and enabling a voltage monitor is most effectively performed by a single function. The API function configure performs configuration and enables the monitor in order to properly enforce the timing and register write ordering constraints.

The LVD driver configures all of the settings of the available configurable LVD monitors.

The settings include:

- voltage_threshold: Determines the voltage detection threshold (i.e. 2.99 Volts).
- sample_clock_divisor: Determines the sample clock rate of the digital filter, based on division of the LOCO clock. Also disables or enables the digital filter if available on the MCU.
- detection_response: Determines which event will occur, reset, interrupt, non-maskable interrupt, or no response, when the voltage threshold is crossed
- voltage_slope: Choose either rising or falling voltage as the trigger for a voltage detection interrupt.
- negation_delay: Determine whether timing of the negation of the voltage event is based upon the reset event or based on the voltage event itself.
- p_callback: Address of user defined function to be called when the voltage event interrupt occurs.

Note

Low Voltage Monitor 0 (LVD0) is not configurable at runtime but can be configured by changing the OFS1 register value on the BSP Properties tab of the Synergy Project Configurator in the e2 studio ISDE.
Digital filter is not to be used with standby modes. If software standby or deep standby mode is to be used, the

digital filter should be disabled.

For details about the implementation of the driver functions see section LVD.

Macro Definition Documentation

◆ LVD_API_VERSION_MAJOR

#define LVD_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ lvd_ctrl_t

typedef void lvd_ctrl_t
LVD control block. Allocate an instance specific control block to pass into the LVD API calls. **Implemented as** ◦ lvd_instance_ctrl_t

Enumeration Type Documentation

◆ lvd_current_state_t

enum lvd_current_state_t	
Instantaneous status of VCC (above or below threshold)	
Enumerator	
LVD_CURRENT_STATE_BELOW_THRESHOLD	VCC < threshold.
LVD_CURRENT_STATE_ABOVE_THRESHOLD	VCC >= threshold or monitor is disabled.

◆ lvd_response_t

enum lvd_response_t	
Response types to a threshold crossing event, interrupt, reset, NMI...	
Enumerator	
LVD_RESPONSE_NMI	Non-maskable interrupt.
LVD_RESPONSE_INTERRUPT	Maskable interrupt.
LVD_RESPONSE_RESET	Reset.
LVD_RESPONSE_NONE	No response, status must be requested via statusGet function.

◆ lvd_threshold_crossing_t

enum lvd_threshold_crossing_t	
Threshold crossing detection (latched)	
Enumerator	
LVD_THRESHOLD_CROSSING_NOT_DETECTED	Threshold crossing has not been detected.
LVD_THRESHOLD_CROSSING_DETECTED	Threshold crossing has been detected.

◆ lvd_threshold_t

enum lvd_threshold_t	
Voltage detection level The thresholds supported by each MCU is in the MCU User's Manual as well as in the r_lvd module description on the threads tab of the Synergy project.	
Enumerator	
LVD_THRESHOLD_MONITOR_1_LEVEL_0	4.29V (Vdet1_0)
LVD_THRESHOLD_MONITOR_1_LEVEL_1	4.14V (Vdet1_1)
LVD_THRESHOLD_MONITOR_1_LEVEL_2	4.02V (Vdet1_2)
LVD_THRESHOLD_MONITOR_1_LEVEL_3	3.84V (Vdet1_3)
LVD_THRESHOLD_MONITOR_1_LEVEL_4	3.10V (Vdet1_4)
LVD_THRESHOLD_MONITOR_1_LEVEL_5	3.00V (Vdet1_5)

LVD_THRESHOLD_MONITOR_1_LEVEL_6	2.90V (Vdet1_6)
LVD_THRESHOLD_MONITOR_1_LEVEL_7	2.79V (Vdet1_7)
LVD_THRESHOLD_MONITOR_1_LEVEL_8	2.68V (Vdet1_8)
LVD_THRESHOLD_MONITOR_1_LEVEL_9	2.58V (Vdet1_9)
LVD_THRESHOLD_MONITOR_1_LEVEL_A	2.48V (Vdet1_A)
LVD_THRESHOLD_MONITOR_1_LEVEL_B	2.20V (Vdet1_B)
LVD_THRESHOLD_MONITOR_1_LEVEL_C	1.96V (Vdet1_C)
LVD_THRESHOLD_MONITOR_1_LEVEL_D	1.86V (Vdet1_D)
LVD_THRESHOLD_MONITOR_1_LEVEL_E	1.75V (Vdet1_E)
LVD_THRESHOLD_MONITOR_1_LEVEL_F	1.65V (Vdet1_F)
LVD_THRESHOLD_MONITOR_1_LEVEL_11	2.99V (Vdet1_11)
LVD_THRESHOLD_MONITOR_1_LEVEL_12	2.92V (Vdet1_12)
LVD_THRESHOLD_MONITOR_1_LEVEL_13	2.85V (Vdet1_13)
LVD_THRESHOLD_MONITOR_2_LEVEL_0	4.29V (Vdet2_0)
LVD_THRESHOLD_MONITOR_2_LEVEL_1	4.14V (Vdet2_1)
LVD_THRESHOLD_MONITOR_2_LEVEL_2	4.02V (Vdet2_2)
LVD_THRESHOLD_MONITOR_2_LEVEL_3	3.84V (Vdet2_3)
LVD_THRESHOLD_MONITOR_2_LEVEL_5	2.99V (Vdet2_5)
LVD_THRESHOLD_MONITOR_2_LEVEL_6	2.92V (Vdet2_6)
LVD_THRESHOLD_MONITOR_2_LEVEL_7	2.85V (Vdet2_7)

◆ lvd_voltage_slope_t

enum lvd_voltage_slope_t	
Voltage slope, rising, falling, or both	
Enumerator	
LVD_VOLTAGE_SLOPE_RISING	When VCC >= Vdet2 (rise) is detected.
LVD_VOLTAGE_SLOPE_FALLING	When VCC < Vdet2 (drop) is detected.
LVD_VOLTAGE_SLOPE_BOTH	When drop and rise are detected.

lvd_status_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Voltage Detection Interface

#include <r_lvd_api.h>

Data Fields

lvd_threshold_crossing_t	crossing_detected
lvd_current_state_t	current_state

Detailed Description

Voltage monitor status structure, used with statusGet function and p_callback to provide current state of the monitor, (threshold crossing detected, vcc currently within range).

Field Documentation

◆ crossing_detected

lvd_threshold_crossing_t lvd_status_t::crossing_detected
Threshold crossing detection (latched)

◆ current_state

lvd_current_state_t lvd_status_t::current_state
Instantaneous status of monitored voltage (above or below threshold)

The documentation for this struct was generated from the following file:

- r_lvd_api.h

lvd_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Voltage Detection Interface

#include <r_lvd_api.h>

Data Fields

uint32_t	monitor_number Monitor number.
lvd_status_t	status Status of monitor.
void const *	p_context Placeholder for user data.

Detailed Description

LVD callback parameter definition

The documentation for this struct was generated from the following file:

- r_lvd_api.h

lvd_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Voltage Detection Interface

#include <r_lvd_api.h>

Data Fields

const uint32_t	monitor_number
lvd_threshold_t	voltage_threshold
lvd_response_t	detection_response
lvd_voltage_slope_t	voltage_slope
uint8_t	monitor_ipl
void(*	p_callback)(lvd_callback_args_t *p_args)
void const *	p_context
void const *	p_extend

Detailed Description

LVD configuration structure

Field Documentation

◆ detection_response

lvd_response_t lvd_cfg_t::detection_response
Response on detecting a threshold crossing

◆ monitor_ipl

uint8_t lvd_cfg_t::monitor_ipl
Interrupt priority level.

◆ monitor_number

const uint32_t lvd_cfg_t::monitor_number
Monitor number, 1, 2, ...

◆ p_callback

void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args)
User function to be called from interrupt

◆ p_context

void const* lvd_cfg_t::p_context
Placeholder for user data. Passed to the user callback in

◆ p_extend

void const* lvd_cfg_t::p_extend
Extension parameter for hardware specific settings

◆ voltage_slope

lvd_voltage_slope_t lvd_cfg_t::voltage_slope
Rising or falling voltage is to be detected

◆ voltage_threshold

lvd_threshold_t lvd_cfg_t::voltage_threshold
Threshold for out of range voltage detection

The documentation for this struct was generated from the following file:

- r_lvd_api.h

lvd_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Voltage Detection Interface

#include <r_lvd_api.h>

Data Fields

ssp_err_t(*	open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
ssp_err_t(*	statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
ssp_err_t(*	statusClear)(lvd_ctrl_t *const p_ctrl)

ssp_err_t(*	close)(lvd_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API.

Field Documentation

◆ close

ssp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)

Disables the LVD peripheral. Closes the driver instance.

Implemented as

- R_LVD_Close()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ open

ssp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.

Implemented as

- R_LVD_Open()

Parameters

[in]	p_ctrl	Pointer to monitor control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

◆ statusClear

ssp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)

Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

Implemented as

- R_LVD_StatusClear()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ statusGet

ssp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)

Get the current state of the monitor, (threshold crossing detected, voltage currently within range) Can be used to poll the state of the LVD monitor at any time. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

Implemented as

- R_LVD_StatusGet()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance
[in,out]	p_lvd_status	Pointer to an lvd_status_t instance

◆ versionGet

ssp_err_t(* lvd_api_t::versionGet) (ssp_version_t *const p_version)

Returns the LVD driver version based on compile time macros.

Implemented as

- R_LVD_VersionGet()

Parameters

[in,out]	p_version	Pointer to version structure

The documentation for this struct was generated from the following file:

- r_lvd_api.h

lvd_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Low Voltage Detection Interface

#include <r_lvd_api.h>

Data Fields

lvd_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
lvd_cfg_t const *	p_cfg Pointer to the configuration structure for this interface instance.
lvd_api_t const *	p_api Pointer to the API structure for this interface instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_lvd_api.h

5.1.4.26 OPAMP Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for Operational Amplifiers. More...

Data Structures

struct	opamp_trim_args_t
struct	opamp_info_t
struct	opamp_status_t
struct	opamp_cfg_t
struct	opamp_api_t
struct	opamp_instance_t

Macros

#define	OPAMP_API_VERSION_MAJOR (2U)

Typedefs

typedef void	opamp_ctrl_t

Enumerations

enum	opamp_trim_cmd_t { OPAMP_TRIM_CMD_START, OPAMP_TRIM_CMD_NEXT_STEP, OPAMP_TRIM_CMD_CLEAR_BIT }
enum	opamp_trim_input_t { OPAMP_TRIM_INPUT_PCH = 0U, OPAMP_TRIM_INPUT_NCH = 1U }

Detailed Description

Interface for Operational Amplifiers.

Summary

The OPAMP interface provides standard operational amplifier functionality, including starting and stopping the amplifier.

Implemented by: Operational Amplifier (OPAMP)

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

OPAMP Interface description: OPAMP Driver

Macro Definition Documentation

◆ OPAMP_API_VERSION_MAJOR

#define OPAMP_API_VERSION_MAJOR (2U)
Includes board and MCU related header files. Version Number of API.

Typedef Documentation

◆ opamp_ctrl_t

typedef void opamp_ctrl_t
OPAMP control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ opamp_trim_cmd_t

enum opamp_trim_cmd_t	
Trim command.	
Enumerator	
OPAMP_TRIM_CMD_START	Initialize trim state machine.
OPAMP_TRIM_CMD_NEXT_STEP	Move to next step in state machine.
OPAMP_TRIM_CMD_CLEAR_BIT	Clear trim bit.

◆ opamp_trim_input_t

enum opamp_trim_input_t	
Trim input.	
Enumerator	
OPAMP_TRIM_INPUT_PCH	Trim non-inverting (+) input.
OPAMP_TRIM_INPUT_NCH	Trim inverting (-) input.

opamp_trim_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

uint8_t	channel Channel.
opamp_trim_input_t	input Which input of the channel above.

Detailed Description

OPAMP trim arguments.

The documentation for this struct was generated from the following file:

- r_opamp_api.h

opamp_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

uint32_t	min_stabilization_wait_us Minimum stabilization wait time in microseconds.

Detailed Description

OPAMP information.

The documentation for this struct was generated from the following file:

- r_opamp_api.h

opamp_status_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

uint32_t	operating_channel_mask Bitmask of channels currently operating.

Detailed Description

OPAMP status.

The documentation for this struct was generated from the following file:

- r_opamp_api.h

opamp_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

OPAMP general configuration.

The documentation for this struct was generated from the following file:

- r_opamp_api.h

opamp_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

ssp_err_t(*	open)(opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)
ssp_err_t(*	start)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
ssp_err_t(*	stop)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
ssp_err_t(*	trim)(opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)
ssp_err_t(*	infoGet)(opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)
ssp_err_t(*	statusGet)(opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)
ssp_err_t(*	close)(opamp_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

OPAMP functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl)

Close the specified OPAMP unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Implemented as

- R_OPAMP_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ infoGet

ssp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

Implemented as

- R_OPAMP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_info	OPAMP information stored here

◆ open

ssp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)

Initialize the operational amplifier.

Implemented as

- R_OPAMP_Open()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ start

ssp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Start the op-amp(s).

Implemented as

- R_OPAMP_Start()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	channel_mask	Bitmask of channels to start

◆ statusGet

ssp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)

Provide status of each op-amp channel.

Implemented as

- R_OPAMP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_status	Status stored here

◆ stop

ssp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Stop the op-amp(s).

Implemented as

- R_OPAMP_Stop()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	channel_mask	Bitmask of channels to stop

◆ trim

ssp_err_t(* opamp_api_t::trim) (opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)

Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details.

Implemented as

- R_OPAMP_Trim()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	cmd	Trim command
[in]	p_args	Pointer to arguments for the command

◆ versionGet

ssp_err_t(* opamp_api_t::versionGet) (ssp_version_t *const p_version)
Retrieve the API version. **Implemented as** ◦ R_OPAMP_VersionGet() **Precondition** This function retrieves the API version. **Parameters**

[in]	p_version	Pointer to version structure

The documentation for this struct was generated from the following file:

- r_opamp_api.h

opamp_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » OPAMP Interface

#include <r_opamp_api.h>

Data Fields

opamp_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
opamp_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
opamp_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_opamp_api.h

5.1.4.27 PDC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for PDC functions. More...

Data Structures

struct	pdc_state_t
struct	pdc_callback_args_t
struct	pdc_cfg_t
struct	pdc_api_t
struct	pdc_instance_t

Macros

#define	PDC_API_VERSION_MAJOR (2U)

Typedefs

typedef void	pdc_ctrl_t

Enumerations

enum	pdc_clock_division_t { PDC_CLOCK_DIVISION_2 = 0u, PDC_CLOCK_DIVISION_4 = 1u, PDC_CLOCK_DIVISION_6 = 2u, PDC_CLOCK_DIVISION_8 = 3u, PDC_CLOCK_DIVISION_10 = 4u, PDC_CLOCK_DIVISION_12 = 5u, PDC_CLOCK_DIVISION_14 = 6u, PDC_CLOCK_DIVISION_16 = 7u }
enum	pdc_endian_t { PDC_ENDIAN_LITTLE = 0u, PDC_ENDIAN_BIG = 1u }
enum	pdc_hsync_polarity_t { PDC_HSYNC_POLARITY_HIGH = 0u, PDC_HSYNC_POLARITY_LOW = 1u }
enum	pdc_vsync_polarity_t { PDC_VSYNC_POLARITY_HIGH = 0u, PDC_VSYNC_POLARITY_LOW = 1u }
enum	pdc_event_t { PDC_EVENT_TRANSFER_COMPLETE = 0u,

	PDC_EVENT_RX_DATA_READY = 0x01u, PDC_EVENT_FRAME_END = 0x02u, PDC_EVENT_ERR_OVERRUN = 0x04u, PDC_EVENT_ERR_UNDERRUN = 0x08u, PDC_EVENT_ERR_V_SET = 0x10u, PDC_EVENT_ERR_H_SET = 0x20u }
enum	pdc_vsync_state_t { PDC_VSYNC_STATE_LOW = 0u, PDC_VSYNC_STATE_HIGH = 1u }
enum	pdc_hsync_state_t { PDC_HSYNC_STATE_LOW = 0u, PDC_HSYNC_STATE_HIGH = 1u }

Detailed Description

Interface for PDC functions.

Summary

The PDC interface provides the functionality for capturing an image from a camera. When a capture is complete a transfer complete interrupt is triggered.

Known Implementations

See also
PDC

Related SSP architecture topics:

- What is an SSP Interface? SSP Interfaces
- What is a SSP Layer? SSP Predefined Layers
- How to use SSP Interfaces and Modules? Using SSP Modules

Macro Definition Documentation

◆ PDC_API_VERSION_MAJOR

#define PDC_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ pdc_ctrl_t

typedef void pdc_ctrl_t
PDC control block. Allocate an instance specific control block to pass into the PDC API calls. **Implemented as** ◦ pdc_instance_ctrl_t

Enumeration Type Documentation

◆ pdc_clock_division_t

enum pdc_clock_division_t	
Clock divider applied to PDC clock to provide PCKO output frequency	
Enumerator	
PDC_CLOCK_DIVISION_2	CLK / 2.
PDC_CLOCK_DIVISION_4	CLK / 4.
PDC_CLOCK_DIVISION_6	CLK / 6.
PDC_CLOCK_DIVISION_8	CLK / 8.
PDC_CLOCK_DIVISION_10	CLK / 10.
PDC_CLOCK_DIVISION_12	CLK / 12.
PDC_CLOCK_DIVISION_14	CLK / 14.
PDC_CLOCK_DIVISION_16	CLK / 16.

◆ pdc_endian_t

enum pdc_endian_t	
Endian of captured data	
Enumerator	
PDC_ENDIAN_LITTLE	Data is in little endian format.
PDC_ENDIAN_BIG	Data is in big endian format.

◆ pdc_event_t

enum pdc_event_t	
PDC events	
Enumerator	
PDC_EVENT_TRANSFER_COMPLETE	Complete frame transferred by DMAC/DTC.
PDC_EVENT_RX_DATA_READY	Receive data ready interrupt.
PDC_EVENT_FRAME_END	Frame end interrupt.
PDC_EVENT_ERR_OVERRUN	Overrun interrupt.
PDC_EVENT_ERR_UNDERRUN	Underrun interrupt.
PDC_EVENT_ERR_V_SET	Vertical line setting error interrupt.
PDC_EVENT_ERR_H_SET	Horizontal byte number setting error interrupt.

◆ pdc_hsync_polarity_t

enum pdc_hsync_polarity_t	
Polarity of input HSYNC signal	
Enumerator	
PDC_HSYNC_POLARITY_HIGH	HSYNC signal is active high.
PDC_HSYNC_POLARITY_LOW	HSYNC signal is active low.

◆ pdc_hsync_state_t

enum pdc_hsync_state_t	
HSYNC signal state	
Enumerator	
PDC_HSYNC_STATE_LOW	HSYNC signal is low.
PDC_HSYNC_STATE_HIGH	HSYNC signal is high.

◆ pdc_vsync_polarity_t

enum pdc_vsync_polarity_t	
Polarity of input VSYNC signal	
Enumerator	
PDC_VSYNC_POLARITY_HIGH	VSYNC signal is active high.
PDC_VSYNC_POLARITY_LOW	VSYNC signal is active low.

◆ pdc_vsync_state_t

enum pdc_vsync_state_t	
VSYNC signal state	
Enumerator	
PDC_VSYNC_STATE_LOW	VSYNC signal is low.
PDC_VSYNC_STATE_HIGH	VSYNC signal is high.

pdc_state_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PDC Interface

#include <r_pdc_api.h>

Data Fields

pdc_vsync_state_t	vsync VSYNC signal state.
pdc_hsync_state_t	hsync HSYNC signal state.

Detailed Description

PDC VSYNC/HSYNC state

The documentation for this struct was generated from the following file:

- r_pdc_api.h

pdc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PDC Interface

#include <r_pdc_api.h>

Data Fields

pdc_event_t	event Event causing the callback.
uint8_t *	p_buffer Pointer to buffer containing the captured data.
void const *	p_context Placeholder for user data. Set in pdc_api_t::open function in pdc_cfg_t.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_pdc_api.h

pdc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PDC Interface

#include <r_pdc_api.h>

Data Fields

uint16_t	x_capture_start_pixel Horizontal position to start capture.
uint16_t	x_capture_pixels Number of horizontal pixels to capture.
uint16_t	y_capture_start_pixel Vertical position to start capture.
uint16_t	y_capture_pixels Number of vertical lines/pixels to capture.
pdc_clock_division_t	clock_division Clock divider.
pdc_endian_t	endian Endian of capture data.
pdc_hsync_polarity_t	hsync_polarity Polarity of HSYNC input.
pdc_vsync_polarity_t	vsync_polarity Polarity of VSYNC input.
uint8_t *	p_buffer Pointer to buffer to write image into.
uint8_t	bytes_per_pixel Number of bytes per pixel.
uint8_t	frame_end_ipl Frame end interrupt priority.
uint8_t	irq_ipl

	PDC interrupt priority.
transfer_instance_t const *const	p_lower_lvl_transfer Pointer to the transfer instance the PDC should use.
void(*	p_callback)(pdc_callback_args_t *p_args) Callback provided when a PDC transfer ISR occurs.
void const *	p_context
void const *	p_extend

Detailed Description

PDC configuration parameters.

Field Documentation

◆ p_context

void const* pdc_cfg_t::p_context
Placeholder for user data. Passed to the user callback in pdc_callback_args_t.

◆ p_extend

void const* pdc_cfg_t::p_extend
Extension parameter for hardware specific settings.

The documentation for this struct was generated from the following file:

- r_pdc_api.h

pdc_api_t Struct Reference
Renesas Synergy Software Package Reference » HAL Interfaces » PDC Interface

#include <r_pdc_api.h>

Data Fields

ssp_err_t(*	open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
ssp_err_t(*	close)(pdc_ctrl_t *const p_ctrl)
ssp_err_t(*	captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
ssp_err_t(*	stateGet)(pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
ssp_err_t(*	versionGet)(ssp_version_t *const p_data)

Detailed Description

PDC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ captureStart

ssp_err_t(* pdc_api_t::captureStart) (pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

Start a capture.

Implemented as

- R_PDC_CaptureStart()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to store captured image data.

◆ close

ssp_err_t(* pdc_api_t::close) (pdc_ctrl_t *const p_ctrl)

Closes the driver and allows reconfiguration. May reduce power consumption.

Implemented as

- R_PDC_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ open

ssp_err_t(* pdc_api_t::open) (pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_PDC_Open()

Note

To reconfigure after calling this function, call pdc_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ stateGet

ssp_err_t(* pdc_api_t::stateGet) (pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)

Get the state of the VSYNC and HSYNC pins.

Implemented as

- R_PDC_StateGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_state	Pointer to store state data.

◆ versionGet

ssp_err_t(* pdc_api_t::versionGet) (ssp_version_t *const p_data)

Return the version of the driver.

Implemented as

- R_PDC_VersionGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

The documentation for this struct was generated from the following file:

- r_pdc_api.h

pdc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PDC Interface

#include <r_pdc_api.h>

Data Fields

pdc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
pdc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
pdc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_pdc_api.h

5.1.4.28 PTP driver Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for PTP functions. More...

Data Structures

struct	UInt64_t
struct	ptp_timestamp_t
struct	ptp_address_t
union	ptp_announce_flag_t
struct	ptp_clock_quality_t
struct	ptp_announce_message_t
struct	ptp_message_reception_t
struct	ptp_callback_args_t
struct	ptp_cfg_t
struct	ptp_api_t
struct	ptp_instance_t

Typedefs

typedef void	ptp_ctrl_t

Enumerations

enum	ptp_device_t { PTP_DEVICE_DISABLED = 0xFFU, PTP_DEVICE_ORDINARY_CLOCK0 = 0U, PTP_DEVICE_ORDINARY_CLOCK1, PTP_DEVICE_BOUNDARY_CLOCK, PTP_DEVICE_TRANSPARENT_CLOCK }
enum	ptp_delay_mechanism_t { PTP_DELAY_MECHANISM_DISABLED = 0xFFU, PTP_DELAY_MECHANISM_P2P = 0U, PTP_DELAY_MECHANISM_E2E }
enum	ptp_state_t { PTP_STATE_DISABLED = 0xFFU, PTP_STATE_MASTER = 0U, PTP_STATE_SLAVE, PTP_STATE_LISTENING }
enum	ptp_stca_mode_t { PTP_STCA_MODE_1 = 0x00U, PTP_STCA_MODE_2_HW = 0x02U, PTP_STCA_MODE_2_SW = 0x03U }
enum	ptp_frame_format_t { PTP_FRAME_FORMAT_DISABLED = 0xFFU, PTP_FRAME_FORMAT_ETH = 0x00U, PTP_FRAME_FORMAT_ETH_8023, PTP_FRAME_FORMAT_UDP4, PTP_FRAME_FORMAT_UDP4_8023 }

```
enum ptp_stca_timer_channel_t {
  PTP_STCA_TIMER_CHANNEL_0 = 0x01U,
PTP_STCA_TIMER_CHANNEL_1 = 0x02U,
PTP_STCA_TIMER_CHANNEL_2 = 0x04U,
PTP_STCA_TIMER_CHANNEL_3 = 0x08U,
  PTP_STCA_TIMER_CHANNEL_4 = 0x10U,
PTP_STCA_TIMER_CHANNEL_5 = 0x20U
}
```

```
enum ptp_stca_timer_pulse_edge_t {
PTP_STCA_TIMER_PULSE_EDGE_RISING = 0U,
PTP_STCA_TIMER_PULSE_EDGE_FALLING }
```

```
enum ptp_event_t {
  PTP_EVENT_TIMER = 0U, PTP_EVENT_STCA, PTP_EVENT_PRCTC,
PTP_EVENT_SYNFP0,
  PTP_EVENT_SYNFP1
}
```

Detailed Description

Interface for PTP functions.

Summary

The PTP interface provides time synchronization functionality.

The PTP interface can be implemented by:

- PTP

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

PTP Interface description: PTP Driver on r_ptp

Typedef Documentation

◆ ptp_ctrl_t

typedef void ptp_ctrl_t

PTP control block. Allocate an instance specific control block to pass into the PTP API calls.

Implemented as

- ptp_instance_ctrl_t

Enumeration Type Documentation

◆ ptp_delay_mechanism_t

enum ptp_delay_mechanism_t	
PTP Delay correction mechanism definitions	
Enumerator	
PTP_DELAY_MECHANISM_DISABLED	Unsupported.
PTP_DELAY_MECHANISM_P2P	Peer to peer delay mechanism.
PTP_DELAY_MECHANISM_E2E	End to end delay mechanism.

◆ ptp_device_t

enum ptp_device_t	
PTP Clock type definitions	
Enumerator	
PTP_DEVICE_DISABLED	Unsupported.
PTP_DEVICE_ORDINARY_CLOCK0	Ordinary Clock Port 0.
PTP_DEVICE_ORDINARY_CLOCK1	Ordinary Clock Port 1.
PTP_DEVICE_BOUNDARY_CLOCK	Boundary Clock.
PTP_DEVICE_TRANSPARENT_CLOCK	Transparent Clock.

◆ ptp_event_t

enum ptp_event_t	
MINT interrupt register definitions	
Enumerator	
PTP_EVENT_TIMER	Interrupt from Timer.
PTP_EVENT_STCA	Interrupt from STCA.
PTP_EVENT_PRCTC	Interrupt from PRC-TC.
PTP_EVENT_SYNFP0	Interrupt from SYNFP0.
PTP_EVENT_SYNFP1	Interrupt from SYNFP1.

◆ ptp_frame_format_t

enum ptp_frame_format_t	
PTP message frame format definitions	
Enumerator	
PTP_FRAME_FORMAT_DISABLED	Unsupported.
PTP_FRAME_FORMAT_ETH	Ethernet II frame format.
PTP_FRAME_FORMAT_ETH_8023	Ethernet 802.3 frame format.
PTP_FRAME_FORMAT_UDP4	Ethernet II over UDP4 frame format.
PTP_FRAME_FORMAT_UDP4_8023	Ethernet 802.3 over UDP4 frame format.

◆ ptp_state_t

enum ptp_state_t	
PTP clock state definitions	
Enumerator	
PTP_STATE_DISABLED	Unsupported.
PTP_STATE_MASTER	Master state.
PTP_STATE_SLAVE	Slave state.
PTP_STATE_LISTENING	Listening state.

◆ ptp_stca_mode_t

enum ptp_stca_mode_t	
STCA mode and gradient setting definitions	
Enumerator	
PTP_STCA_MODE_1	Mode1 (not use STCA)
PTP_STCA_MODE_2_HW	Mode2 (use STCA) and HW gradient setting.
PTP_STCA_MODE_2_SW	Mode2 (use STCA) and SW gradient setting.

◆ ptp_stca_timer_channel_t

enum ptp_stca_timer_channel_t	
STCA pulse output timer channel definitions	
Enumerator	
PTP_STCA_TIMER_CHANNEL_0	STCA pulse output timer 0.
PTP_STCA_TIMER_CHANNEL_1	STCA pulse output timer 1.
PTP_STCA_TIMER_CHANNEL_2	STCA pulse output timer 2.
PTP_STCA_TIMER_CHANNEL_3	STCA pulse output timer 3.
PTP_STCA_TIMER_CHANNEL_4	STCA pulse output timer 4.
PTP_STCA_TIMER_CHANNEL_5	STCA pulse output timer 5.

◆ ptp_stca_timer_pulse_edge_t

enum ptp_stca_timer_pulse_edge_t	
STCA pulse output timer edge definitions	
Enumerator	
PTP_STCA_TIMER_PULSE_EDGE_RISING	Rising edge.
PTP_STCA_TIMER_PULSE_EDGE_FALLING	Falling edge.

UInt64_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

PTP data type structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_timestamp_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

PTP message timestamp structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_address_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

PTP channel related structure (MAC address, IP address)

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_announce_flag_t Union Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

Announce flagField type structure

The documentation for this union was generated from the following file:

- r_ptp_api.h

ptp_clock_quality_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

PTP clock quality structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_announce_message_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

Announce message field type structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_message_reception_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Detailed Description

PTP message reception configuration field structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Data Fields

void const *	p_context Context provided to user during callback.
ptp_event_t	event The event can be used to identify what caused the callback.
ptp_stca_timer_channel_t	timer_channel STCA pulse output timer channel.

Detailed Description

PTP callback arguments definition

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Data Fields

ssp_err_t(*	p_callback)(ptp_callback_args_t *p_args) Pointer to interrupt callback function.
void const *	p_context User defined context passed into callback function.
uint8_t	irq_ipl MINT interrupt IRQ number.
ptp_device_t	device PTP clock type.
ptp_state_t	state [2] PTP clock state.
ptp_delay_mechanism_t	delay [2] Delay correction mechanism.
ptp_frame_format_t	frame_format [2] PTP message frame format.
ptp_stca_mode_t	stca_sync_mode STCA synchronous mode.
void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

PTP user configuration structure

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Data Fields

ssp_err_t(*	open)(ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)
ssp_err_t(*	close)(ptp_ctrl_t *const p_ctrl)
ssp_err_t(*	configure)(ptp_ctrl_t *const p_ctrl, uint32_t *p_ip_address, uint32_t *p_physical_address_msw, uint32_t *p_physical_address_lsw)
ssp_err_t(*	setExtPromiscuous)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, bool is_set)
ssp_err_t(*	setLocalClock)(ptp_ctrl_t *const p_ctrl, ptp_timestamp_t *p_clock)
ssp_err_t(*	getLocalClock)(ptp_ctrl_t *const p_ctrl, ptp_timestamp_t *p_clock, uint32_t wait_option)
ssp_err_t(*	getMasterPortID)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port)
ssp_err_t(*	setMasterPortID)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port)
ssp_err_t(*	getSyncInfo)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_timeInterval_t *p_master_offset, ptp_timeInterval_t *p_path_delay)
ssp_err_t(*	start)(ptp_ctrl_t *const p_ctrl, uint32_t wait_option)
ssp_err_t(*	stop)(ptp_ctrl_t *const p_ctrl, uint32_t wait_option)
ssp_err_t(*	setWorst10Values)(ptp_ctrl_t *const p_ctrl, uint8_t interval)
ssp_err_t(*	checkWorst10Values)(ptp_ctrl_t *const p_ctrl, uint32_t wait_option)
ssp_err_t(*	getWorst10Values)(ptp_ctrl_t *const p_ctrl, uint32_t *p_positive_worst10, uint32_t *p_negative_worst10, uint32_t wait_option)
ssp_err_t(*	setGradientLimit)(ptp_ctrl_t *const p_ctrl, uint32_t *p_positive_limit, uint32_t *p_negative_limit)

ssp_err_t(*	updateClockID)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_clock_id)
ssp_err_t(*	updateDomainNumber)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint8_t domain_num)
ssp_err_t(*	updateAnnounceFlags)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_announce_flag_t *p_flag)
ssp_err_t(*	updateAnnounceMsgs)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_announce_message_t *p_message)
ssp_err_t(*	updateSyncAnnounceMsgInterval)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_sync_interval, int8_t *p_announce_interval)
ssp_err_t(*	updateDelayMsgInterval)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_interval, uint32_t *p_timeout)
ssp_err_t(*	getMessageReceptionConfig)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception)
ssp_err_t(*	setMessageReceptionConfig)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception)
ssp_err_t(*	disableTimer)(ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel)
ssp_err_t(*	indicateEvent)(ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set)
ssp_err_t(*	autoClearEvent)(ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set)
ssp_err_t(*	setTimer)(ptp_ctrl_t *const p_ctrl, uint8_t timer_channel, UInt64_t event, uint32_t cycle, uint32_t pulse_width)
ssp_err_t(*	setMINTevent)(ptp_ctrl_t *const p_ctrl, ptp_event_t ptp_reg, uint32_t event, bool is_set)
ssp_err_t(*	enableINFABTnotification)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)
ssp_err_t(*	disableINFABTnotification)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)
ssp_err_t(*	checkINFABTstatus)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint8_t *p_status)

ssp_err_t(*	clearINFABTstatus)(ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

PTP functions implemented at the HAL layer follow this API.

Field Documentation

◆ autoClearEvent

ssp_err_t(* ptp_api_t::autoClearEvent) (ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set)

Set/clear auto clear mode for enabling one time output of ELC event.

Implemented as

- R_PTP_AutoClearEvent()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	timer_channel	STCA pulse output timer channel
[in]	timer_pulse_edge	STCA pulse output timer edge
[in]	is_set	Enable or disable automatic clearing of event

◆ checkINFABTstatus

ssp_err_t(* ptp_api_t::checkINFABTstatus) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint8_t *p_status)

Checks the status of INFABT flag

Implemented as

- R_PTP_CheckINFABTstatus()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[out]	p_status	Returns status of INFABT flag

◆ checkWorst10Values

ssp_err_t(* ptp_api_t::checkWorst10Values) (ptp_ctrl_t *const p_ctrl, uint32_t wait_option)

Checks worst 10 values by hardware and set as gradient limits.

Implemented as

- R_PTP_CheckWorst10Values()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	wait_option	Timeout interval

◆ clearINFABTstatus

ssp_err_t(* ptp_api_t::clearINFABTstatus) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)

Clear INFABT interrupt occurrence flag.

Implemented as

- R_PTP_ClearINFABTstatus()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel

◆ close

ssp_err_t(* ptp_api_t::close) (ptp_ctrl_t *const p_ctrl)

Close the PTP driver module.

Implemented as

- R_PTP_Close()

Parameters

[in]	p_ctrl	Pointer to the control structure

◆ configure

ssp_err_t(* ptp_api_t::configure) (ptp_ctrl_t *const p_ctrl, uint32_t *p_ip_address, uint32_t *p_physical_address_msw, uint32_t *p_physical_address_lsw)

Configures the PTP driver module.

Implemented as

- R_PTP_Configure()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	p_ip_address	Pointer to the IP address
[in]	p_physical_address_msw	Pointer to the higher bits of physical address
[in]	p_physical_address_lsw	Pointer to the lower bits of physical address

◆ disableINFABTnotification

ssp_err_t(* ptp_api_t::disableINFABTnotification) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)

Disable EPTPC INFABT notification

Implemented as

- R_PTP_DisableINFABTnotification()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel

◆ disableTimer

ssp_err_t(* ptp_api_t::disableTimer) (ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel)

Disable timer event interrupt.

Implemented as

- R_PTP_DisableTimer()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	timer_channel	STCA pulse output timer channel

◆ enableINFABTnotification

ssp_err_t(* ptp_api_t::enableINFABTnotification) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel)

Enable EPTPC INFABT notification

Implemented as

- R_PTP_EnableINFABTnotification()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel

◆ getLocalClock

ssp_err_t(* ptp_api_t::getLocalClock) (ptp_ctrl_t *const p_ctrl, ptp_timestamp_t *p_clock, uint32_t wait_option)

Gets local clock counter

Implemented as

- R_PTP_GetLocalClock()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	wait_option	Time out interval
[out]	p_clock	Pointer to local clock counter

◆ getMasterPortID

ssp_err_t(* ptp_api_t::getMasterPortID) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port)

Gets master port ID

Implemented as

- R_PTP_GetMasterPortID()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[out]	p_clock	Pointer to local clock counter
[out]	p_port	Pointer to master port

◆ getMessageReceptionConfig

ssp_err_t(* ptp_api_t::getMessageReceptionConfig) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception)

Get message reception configuration

Implemented as

- R_PTP_GetMessageReceptionConfig()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[out]	p_ptp_message_reception	Pointer to SYNFP message reception configuration structure

◆ getSyncInfo

ssp_err_t(* ptp_api_t::getSyncInfo) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_timeInterval_t *p_master_offset, ptp_timeInterval_t *p_path_delay)

Get current offsetFromMaster and meanPathDelay.

Implemented as

- R_PTP_GetSyncInfo()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[out]	p_master_offset	Returns the offset from master
[out]	p_path_delay	Returns the mean path delay

◆ getWorst10Values

ssp_err_t(* ptp_api_t::getWorst10Values) (ptp_ctrl_t *const p_ctrl, uint32_t *p_positive_worst10, uint32_t *p_negative_worst10, uint32_t wait_option)

Get worst 10 values by software.

Implemented as

- R_PTP_GetWorst10Values()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	wait_option	Timeout interval
[out]	p_positive_worst10	Returns the positive worst 10 values
[out]	p_negative_worst10	Returns the negative worst 10 values

◆ indicateEvent

ssp_err_t(* ptp_api_t::indicateEvent) (ptp_ctrl_t *const p_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set)

Set/clear interrupt indication to ELC output on generation of pulse produced by pulse output timer.

Implemented as

- R_PTP_IndicateEvent()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	timer_channel	STCA pulse output timer channel
[in]	timer_pulse_edge	STCA pulse output timer pulse edge
[in]	is_set	Set or clear indication of

◆ open

ssp_err_t(* ptp_api_t::open) (ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)

Open the PTP driver module.

Implemented as

- R_PTP_Open()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	p_cfg	Pointer to a configuration structure

◆ setExtPromiscuous

ssp_err_t(* ptp_api_t::setExtPromiscuous) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, bool is_set)

Sets or clears the extended promiscuous mode

Implemented as

- R_PTP_SetExtPromiscuous()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	is_set	Set/clear extended promiscuous mode

◆ setGradientLimit

ssp_err_t(* ptp_api_t::setGradientLimit) (ptp_ctrl_t *const p_ctrl, uint32_t *p_positive_limit, uint32_t *p_negative_limit)

Set the gradient limits for positive and negative worst 10 values.

Implemented as

- R_PTP_SetGradientLimit()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	p_positive_limit	Positive limit of worst 10 values
[in]	p_negative_limit	Negative limit of worst 10 values

◆ setLocalClock

ssp_err_t(* ptp_api_t::setLocalClock) (ptp_ctrl_t *const p_ctrl, ptp_timestamp_t *p_clock)

Sets local clock counter

Implemented as

- R_PTP_SetLocalClock()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	p_clock	Pointer to local clock counter

◆ setMasterPortID

ssp_err_t(* ptp_api_t::setMasterPortID) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port)

Sets master port ID

Implemented as

- R_PTP_SetMasterPortID()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_clock	Pointer to local clock counter
[in]	p_port	Pointer to master port

◆ setMessageReceptionConfig

ssp_err_t(* ptp_api_t::setMessageReceptionConfig) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception)

Set message reception configuration.

Implemented as

- R_PTP_SetMessageReceptionConfig()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_ptp_message_reception	Pointer to SYNFP message reception config structure

◆ setMINTevent

ssp_err_t(* ptp_api_t::setMINTevent) (ptp_ctrl_t *const p_ctrl, ptp_event_t ptp_reg, uint32_t event, bool is_set)

Set MINT interrupt event to enable notification for change in state of modules.

Implemented as

- R_PTP_SetMINTevent()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_reg	MINT interrupt register
[in]	event	Interrupt element
[in]	is_set	Set or clear MINT event

◆ setTimer

ssp_err_t(* ptp_api_t::setTimer) (ptp_ctrl_t *const p_ctrl, uint8_t timer_channel, UInt64_t event, uint32_t cycle, uint32_t pulse_width)

Sets start time, pulse period and pulse width for the pulse output timer.

Implemented as

- R_PTP_SetTimer()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	timer_channel	STCA pulse output timer channel
[in]	event	Timer event start time
[in]	cycle	Pulse output cycle interval
[in]	pulse_width	Width of the high level of the pulse signal

◆ setWorst10Values

ssp_err_t(* ptp_api_t::setWorst10Values) (ptp_ctrl_t *const p_ctrl, uint8_t interval)

Sets the time interval for collecting worst 10 values

Implemented as

- R_PTP_SetWorst10Values()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	interval	Time interval to collect worst 10 values

◆ start

ssp_err_t(* ptp_api_t::start) (ptp_ctrl_t *const p_ctrl, uint32_t wait_option)

Starts the time synchronization.

Implemented as

- R_PTP_Start()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	wait_option	Timeout interval

◆ stop

ssp_err_t(* ptp_api_t::stop) (ptp_ctrl_t *const p_ctrl, uint32_t wait_option)

Stops the time synchronization.

Implemented as

- R_PTP_Stop()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	wait_option	Timeout interval

◆ updateAnnounceFlags

ssp_err_t(* ptp_api_t::updateAnnounceFlags) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_announce_flag_t *p_flag)

Update the announce message's flag field.

Implemented as

- R_PTP_UpdateAnnounceFlags()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_flag	Pointer to announce message flag field

◆ updateAnnounceMsgs

ssp_err_t(* ptp_api_t::updateAnnounceMsgs) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, ptp_announce_message_t *p_message)

Update the announce message's message field.

Implemented as

- R_PTP_UpdateAnnounceMsgs()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_message	Pointer to announce message field

◆ updateClockID

ssp_err_t(* ptp_api_t::updateClockID) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_clock_id)

Update the clock identity field.

Implemented as

- R_PTP_UpdateClockID()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_clock_id	Pointer to clock ID

◆ updateDelayMsgInterval

ssp_err_t(* ptp_api_t::updateDelayMsgInterval) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_interval, uint32_t *p_timeout)

Update transmission interval, logMessageInterval and timeout values of Delay messages.

Implemented as

- R_PTP_UpdateDelayMsgInterval()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_interval	Case of master: Delay_Resp logMessageInterval Case of slave: Delay_Req/Pdelay_Req transmission interval
[in]	p_timeout	Delay_Resp/Pdelay_Resp receiving timeout

◆ updateDomainNumber

ssp_err_t(* ptp_api_t::updateDomainNumber) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, uint8_t domain_num)

Update the domain number field in the message header.

Implemented as

- R_PTP_UpdateDomainNumber()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	domain_num	Domain number

◆ updateSyncAnnounceMsgInterval

ssp_err_t(* ptp_api_t::updateSyncAnnounceMsgInterval) (ptp_ctrl_t *const p_ctrl, uint8_t ptp_channel, int8_t *p_sync_interval, int8_t *p_announce_interval)

Update transmission interval and logMessageInterval of Sync and Announce messages.

Implemented as

- R_PTP_UpdateSyncAnnounceMsgInterval()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	ptp_channel	EPTPC channel
[in]	p_sync_interval	Pointer to sync message interval
[in]	p_announce_interval	Pointer to announce message interval

◆ versionGet

ssp_err_t(* ptp_api_t::versionGet) (ssp_version_t *const p_version)
Get the driver version based on compile time macros. **Implemented as** ◦ R_PTP_VersionGet() **Parameters** [out] p_version — Code and API version used.

The documentation for this struct was generated from the following file:

- r_ptp_api.h

ptp_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTP driver Interface

#include <r_ptp_api.h>

Data Fields

ptp_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
ptp_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
ptp_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_ptp_api.h

5.1.4.29 PTPEDMAC driver Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for PTPEDMAC functions. More...

Data Structures

struct	ptpedmac_descriptor_t
struct	ptpedmac_callback_args_t
struct	ptpedmac_cfg_t
struct	ptpedmac_api_t
struct	ptpedmac_instance_t

Typedefs

typedef void	ptpedmac_ctrl_t

Enumerations

enum	ptpedmac_event_t { PTPEDMAC_EVENT_READ = 0, PTPEDMAC_EVENT_WRITE, PTPEDMAC_EVENT_ERR }
enum	ptpedmac_trans_t { PTPEDMAC_TRANS_FLAG_OFF = 0, PTPEDMAC_TRANS_FLAG_ON = 1 }

Detailed Description

Interface for PTPEDMAC functions.

Summary

The PTPEDMAC interface supports PTP host interface to receive PTP message.

The PTPEDMAC interface can be implemented by:

- PTPEDMAC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

PTPEDMAC Interface description: PTPEDMAC Driver on r_ptpedmac

Typedef Documentation

◆ ptpedmac_ctrl_t

typedef void ptpedmac_ctrl_t
PTPEDMAC control block. Allocate an instance specific control block to pass into the PTPEDMAC API calls. **Implemented as** ◦ ptpedmac_instance_ctrl_t

Enumeration Type Documentation

◆ ptpedmac_event_t

enum ptpedmac_event_t	
PTPEDMAC interrupt event definitions	
Enumerator	
PTPEDMAC_EVENT_READ	Frame reception interrupt (FR)
PTPEDMAC_EVENT_WRITE	Frame transmission interrupt (TC)
PTPEDMAC_EVENT_ERR	Error interrupt (MACE, RFOF, RDE, TFUF, TDE, ADE and RFCOF)

◆ ptpedmac_trans_t

enum ptpedmac_trans_t	
Enumeration to specify the status of transfer flag	
Enumerator	
PTPEDMAC_TRANS_FLAG_OFF	Transfer disable.
PTPEDMAC_TRANS_FLAG_ON	Transfer enable.

ptpedmac_descriptor_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTPEDMAC driver Interface

#include <r_ptpedmac_api.h>

Detailed Description

PTPEDMAC descriptor structure

The documentation for this struct was generated from the following file:

- r_ptpedmac_api.h

ptpedmac_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTPEDMAC driver Interface

#include <r_ptpedmac_api.h>

Data Fields

uint16_t	channel Device channel number.
uint32_t	ether_frame_type Ethernet PTP message type.
void const *	p_context Context provided to user during callback.
ptpedmac_event_t	event The event can be used to identify what caused the callback.

Detailed Description

PTPEDMAC callback arguments definition

The documentation for this struct was generated from the following file:

- r_ptpedmac_api.h

ptpedmac_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTPEDMAC driver Interface

#include <r_ptpedmac_api.h>

Data Fields

ssp_err_t(*	p_callback)(ptpedmac_callback_args_t *p_args) Pointer to interrupt callback function.
void const *	p_context User defined context passed into callback function.
uint8_t	irq_ipl PINT interrupt IRQ number.

Detailed Description

PTPEDMAC configuration block. Allocate an instance specific control block to pass into the PTPEDMAC API calls.

The documentation for this struct was generated from the following file:

- r_ptpedmac_api.h

ptpedmac_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTPEDMAC driver Interface

#include <r_ptpedmac_api.h>

Data Fields

ssp_err_t(*	open)(ptpedmac_ctrl_t *const p_ctrl, ptpedmac_cfg_t const *const p_cfg)
ssp_err_t(*	linkProcess)(ptpedmac_ctrl_t *const p_api_ctrl)
ssp_err_t(*	linkCheck)(ptpedmac_ctrl_t *const p_ctrl)
ssp_err_t(*	read)(ptpedmac_ctrl_t *const p_ctrl, uint32_t *p_channel, void *const p_buffer, int32_t *p_num_received)
ssp_err_t(*	close)(ptpedmac_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

PTPEDMAC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* ptpedmac_api_t::close) (ptpedmac_ctrl_t *const p_ctrl)

Close the PTPEDMAC driver module.

Implemented as

- R_PTPEDMAC_Close()

Parameters

[in]	p_ctrl	Pointer to the control structure

◆ linkCheck

ssp_err_t(* ptpedmac_api_t::linkCheck) (ptpedmac_ctrl_t *const p_ctrl)

Checks host interface communication status

Implemented as

- R_PTPEDMAC_CheckLink()

Parameters

[in]	p_ctrl	Pointer to the control structure

◆ linkProcess

ssp_err_t(* ptpedmac_api_t::linkProcess) (ptpedmac_ctrl_t *const p_api_ctrl)

Sets host interface to transfer PTP messages

Implemented as

- R_PTPEDMAC_LinkProcess()

Parameters

[in]	p_ctrl	Pointer to the control structure

◆ open

ssp_err_t(* ptpedmac_api_t::open) (ptpedmac_ctrl_t *const p_ctrl, ptpedmac_cfg_t const *const p_cfg)

Open the PTPEDMAC driver module for reception of PTP messages.

Implemented as

- R_PTPEDMAC_Open()

Parameters

[in]	p_ctrl	Pointer to the control structure
[in]	p_cfg	Pointer to configuration structure

◆ read

ssp_err_t(* ptpedmac_api_t::read) (ptpedmac_ctrl_t *const p_ctrl, uint32_t *p_channel, void *const p_buffer, int32_t *p_num_received)

Receives PTP message

Implemented as

- R_PTPEDMAC_Read()

Parameters

[in]	p_ctrl	Pointer to the control structure
[out]	p_channel	Pointer to received channel
[out]	p_buffer	Pointer to received data buffer
[out]	p_num_received	Pointer to number of received data

◆ versionGet

ssp_err_t(* ptpedmac_api_t::versionGet) (ssp_version_t *const p_version)

Get the driver version based on compile time macros.

Implemented as

- R_PTPEDMAC_VersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_ptpedmac_api.h

ptpedmac_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » PTPEDMAC driver Interface

#include <r_ptpedmac_api.h>

Data Fields

ptpedmac_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
ptpedmac_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
ptpedmac_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_ptpedmac_api.h

5.1.4.30 Quad SPI Flash Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for accessing external SPI flash devices. More...

Data Structures

struct	qspi_cfg_t
struct	qspi_info_t
struct	qspi_api_t
struct	qspi_instance_t

Typedefs

typedef void	qspi_ctrl_t

Enumerations

enum	qspi_address_mode_t

Detailed Description

Interface for accessing external SPI flash devices.

Summary

The QSPI module provides an interface that writes and erases sectors in quad SPI flash devices connected to the QSPI interface.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

QSPI Interface description: QSPI Driver

Typedef Documentation

◆ qspi_ctrl_t

typedef void qspi_ctrl_t
QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. **Implemented as** ◦ qspi_instance_ctrl_t

Enumeration Type Documentation

◆ qspi_address_mode_t

enum qspi_address_mode_t
User configuration structure used for selecting addressing mode

qspi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Quad SPI Flash Interface

#include <r_qspi_api.h>

Data Fields

void *	p_extend place holder for future development

Detailed Description

User configuration structure used by the open function

The documentation for this struct was generated from the following file:

- r_qspi_api.h

qspi_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Quad SPI Flash Interface

#include <r_qspi_api.h>

Data Fields

uint32_t	total_size_bytes Size of this QSPI in bytes.
uint32_t	min_program_size_bytes Minimum program size in bytes.
uint32_t *	p_erase_sizes_bytes Array of available erase sizes. Each entry is in bytes.
uint8_t	num_erase_sizes Number of available erase sizes.

Detailed Description

QSPI information structure to store information specific to the currently used QSPI.

The documentation for this struct was generated from the following file:

- r_qspi_api.h

qspi_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Quad SPI Flash Interface

#include <r_qspi_api.h>

Data Fields

ssp_err_t(*	open)(qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
ssp_err_t(*	close)(qspi_ctrl_t *p_ctrl)
ssp_err_t(*	read)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
ssp_err_t(*	pageProgram)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
ssp_err_t(*	erase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint32_t byte_count)
ssp_err_t(*	infoGet)(qspi_ctrl_t *const p_ctrl, qspi_info_t *const p_info)
ssp_err_t(*	sectorErase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
ssp_err_t(*	statusGet)(qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
ssp_err_t(*	bankSelect)(uint32_t bank)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

QSPI functions implemented at the HAL layer follow this API.

Field Documentation

◆ bankSelect

ssp_err_t(* qspi_api_t::bankSelect) (uint32_t bank)

Select the bank to access.

Implemented as

- R_QSPI_BankSelect()

Parameters

[in]	bank	The bank number

◆ close

ssp_err_t(* qspi_api_t::close) (qspi_ctrl_t *p_ctrl)

Close the QSPI driver module.

Implemented as

- R_QSPI_Close()

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ erase

ssp_err_t(* qspi_api_t::erase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint32_t byte_count)

Erase a certain number of bytes of the flash.

Implemented as

- R_QSPI_Erase()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to start the erase from
[in]	byte_count	The number of bytes to erase

◆ infoGet

ssp_err_t(* qspi_api_t::infoGet) (qspi_ctrl_t *const p_ctrl, qspi_info_t *const p_info)

Get information about this specific QSPI flash.

Implemented as

- R_QSPI_InfoGet()

Parameters

[in]	p_ctrl	Control block set in qspi_api_t::open call for this timer.
[out]	p_info	Collection of information for this QSPI.

◆ open

ssp_err_t(* qspi_api_t::open) (qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)

Open the QSPI driver module.

Implemented as

- R_QSPI_Open()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_cfg	Pointer to a configuration structure

◆ pageProgram

ssp_err_t(* qspi_api_t::pageProgram) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)

Program a page of data to the flash.

Implemented as

- R_QSPI_PageProgram()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to write the data to
[in]	p_memory_address	The memory address of the data to write to the flash device
[in]	byte_count	The number of bytes to write

◆ read

ssp_err_t(* qspi_api_t::read) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)

Read a block of data from the flash.

Implemented as

- R_QSPI_Read()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to read
[in]	p_memory_address	The memory address of a buffer to place the read data in
[in]	byte_count	The number of bytes to read

◆ sectorErase

ssp_err_t(* qspi_api_t::sectorErase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)

Erase a sector of the flash.

Implemented as

- R_QSPI_SectorErase()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to start the erase from

◆ statusGet

ssp_err_t(* qspi_api_t::statusGet) (qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)

Get the write or erase status of the flash.

Implemented as

- R_QSPI_StatusGet()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_write_in_progress	The write or erase status - TRUE = write/erase in progress

◆ versionGet

ssp_err_t(* qspi_api_t::versionGet) (ssp_version_t *const p_version)

Get the driver version based on compile time macros.

Implemented as

- R_QSPI_VersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_qspi_api.h

qspi_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Quad SPI Flash Interface

#include <r_qspi_api.h>

Data Fields

qspi_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
qspi_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
qspi_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_qspi_api.h

5.1.4.31 RTC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for accessing the Realtime Clock. More...

Data Structures

struct	rtc_callback_args_t

struct	rtc_error_adjustment_cfg_t
struct	rtc_error_adjustment_mode_cfg_t
struct	rtc_alarm_time_t
struct	rtc_info_t
struct	rtc_cfg_t
struct	rtc_api_t
struct	rtc_instance_t

Macros

#define	RTC_API_VERSION_MAJOR (2U)

Typedefs

typedef struct tm	rtc_time_t
typedef void	rtc_ctrl_t

Enumerations

enum	rtc_event_t { RTC_EVENT_ALARM_IRQ, RTC_EVENT_PERIODIC_IRQ, RTC_EVENT_CARRY_IRQ }
enum	rtc_clock_source_t { RTC_CLOCK_SOURCE_SUBCLK = 0, RTC_CLOCK_SOURCE_LOCO = 1 }
enum	rtc_status_t { RTC_STATUS_STOPPED = 0, RTC_STATUS_RUNNING = 1 }
enum	rtc_error_adjustment_t { RTC_ERROR_ADJUSTMENT_NONE = 0, RTC_ERROR_ADJUSTMENT_ADD_PRESCALER = 1, RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER = 2 }
enum	rtc_error_adjustment_mode_t { RTC_ERROR_ADJUSTMENT_MODE_MANUAL = 0, RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC = 1 }
enum	rtc_error_adjustment_period_t { RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE = 0, RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND = 1, RTC_ERROR_ADJUSTMENT_PERIOD_NONE = 2 }
enum	rtc_periodic_irq_select_t { RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND = 6, RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND,

```
                                    RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND,
                                    RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND,
                                     RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND,
                                    RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND,
                                    RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND,
                                    RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND,
                                     RTC_PERIODIC_IRQ_SELECT_1_SECOND,
                                    RTC_PERIODIC_IRQ_SELECT_2_SECONDS
                                    }
```

Detailed Description

Interface for accessing the Realtime Clock.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

RTC description: RTC Driver

Macro Definition Documentation

◆ RTC_API_VERSION_MAJOR

```
#define RTC_API_VERSION_MAJOR   (2U)
```

Use of time structure, tm

Typedef Documentation

◆ rtc_ctrl_t

```
typedef void rtc_ctrl_t
```

RTC control block. Allocate an instance specific control block to pass into the RTC API calls.

Implemented as

- rtc_instance_ctrl_t

◆ rtc_time_t

```
typedef struct tm rtc_time_t
```

Date and time structure defined in C standard library <time.h>

Enumeration Type Documentation

◆ rtc_clock_source_t

enum rtc_clock_source_t	
Clock source for the RTC block	
Enumerator	
RTC_CLOCK_SOURCE_SUBCLK	Sub-clock oscillator.
RTC_CLOCK_SOURCE_LOCO	Low power On Chip Oscillator.

◆ rtc_error_adjustment_mode_t

enum rtc_error_adjustment_mode_t	
Time error adjustment mode settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_MODE_MANUAL	Adjustment mode is set to manual.
RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC	Adjustment mode is set to automatic.

◆ rtc_error_adjustment_period_t

enum rtc_error_adjustment_period_t	
Time error adjustment period settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE	Adjustment period is set to every one minute.
RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND	Adjustment period is set to every ten second.
RTC_ERROR_ADJUSTMENT_PERIOD_NONE	Adjustment period not supported in manual mode.

◆ rtc_error_adjustment_t

enum rtc_error_adjustment_t	
Time error adjustment settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_NONE	Adjustment is not performed.
RTC_ERROR_ADJUSTMENT_ADD_PRESCALER	Adjustment is performed by the addition to the prescaler.
RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER	Adjustment is performed by the subtraction from the prescaler.

◆ rtc_event_t

enum rtc_event_t	
Events that can trigger a callback function	
Enumerator	
RTC_EVENT_ALARM_IRQ	Real Time Clock ALARM IRQ.
RTC_EVENT_PERIODIC_IRQ	Real Time Clock PERIODIC IRQ.
RTC_EVENT_CARRY_IRQ	Real Time Clock CARRY IRQ.

◆ rtc_periodic_irq_select_t

enum rtc_periodic_irq_select_t	
Periodic Interrupt select	
Enumerator	
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND	A periodic irq is generated every 1/256 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND	A periodic irq is generated every 1/128 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND	A periodic irq is generated every 1/64 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND	A periodic irq is generated every 1/32 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND	A periodic irq is generated every 1/16 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND	A periodic irq is generated every 1/8 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND	A periodic irq is generated every 1/4 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND	A periodic irq is generated every 1/2 second.
RTC_PERIODIC_IRQ_SELECT_1_SECOND	A periodic irq is generated every 1 second.
RTC_PERIODIC_IRQ_SELECT_2_SECONDS	A periodic irq is generated every 2 seconds.

◆ rtc_status_t

enum rtc_status_t	
RTC run state	
Enumerator	
RTC_STATUS_STOPPED	RTC counter is stopped.
RTC_STATUS_RUNNING	RTC counter is running.

rtc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_event_t	event The event can be used to identify what caused the callback (compare match or error).
void const *	p_context Placeholder for user data. Set in r_timer_t::open function in timer_cfg_t.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_error_adjustment_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_error_adjustment_t	adjustment_type Time error adjustment type setting.
uint8_t	adjustment_value Time error adjustment value.

Detailed Description

Time error adjustment value configuration

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_error_adjustment_mode_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_error_adjustment_mode_t	adjustment_mode Time error adjustment mode setting.
rtc_error_adjustment_period_t	adjustment_period Time error adjustment period setting.

Detailed Description

Time error adjustment mode and period configuration

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_alarm_time_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_time_t	time Time structure.

bool	sec_match Enable the alarm based on a match of the seconds field.
bool	min_match Enable the alarm based on a match of the minutes field.
bool	hour_match Enable the alarm based on a match of the hours field.
bool	mday_match Enable the alarm based on a match of the days field.
bool	mon_match Enable the alarm based on a match of the months field.
bool	year_match Enable the alarm based on a match of the years field.
bool	dayofweek_match Enable the alarm based on a match of the dayofweek field.

Detailed Description

Alarm time setting structure

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_clock_source_t	clock_source Clock source for the RTC block.
rtc_status_t	status RTC run status.

Detailed Description

RTC Information Structure for information returned by infoGet()

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_clock_source_t	clock_source Clock source for the RTC block.
bool	hw_cfg Initialize RTC in Open()
uint32_t	error_adjustment_value Value of the prescaler for error adjustment.
rtc_error_adjustment_t	error_adjustment_type How the prescaler value is applied.
uint8_t	alarm_ipl

	Alarm interrupt priority.
uint8_t	periodic_ipl Periodic interrupt priority.
uint8_t	carry_ipl Carry interrupt priority.
void(*	p_callback)(rtc_callback_args_t *p_args) Called from the ISR.
void const *	p_context Passed to the callback.
void const *	p_extend RTC hardware dependant configuration.

Detailed Description

User configuration structure, used in open function

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

ssp_err_t(*	open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
ssp_err_t(*	close)(rtc_ctrl_t *const p_ctrl)

ssp_err_t(*	configure)(rtc_ctrl_t *const p_ctrl, void *const p_extend)
ssp_err_t(*	calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)
ssp_err_t(*	calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)
ssp_err_t(*	calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)
ssp_err_t(*	calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)
ssp_err_t(*	calendarCounterStart)(rtc_ctrl_t *const p_ctrl)
ssp_err_t(*	calendarCounterStop)(rtc_ctrl_t *const p_ctrl)
ssp_err_t(*	irqEnable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
ssp_err_t(*	irqDisable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
ssp_err_t(*	periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)
ssp_err_t(*	infoGet)(rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
ssp_err_t(*	errorAdjustmentModeSet)(rtc_ctrl_t *p_ctrl, rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode)
ssp_err_t(*	errorAdjustmentSet)(rtc_ctrl_t *p_ctrl, rtc_error_adjustment_cfg_t *p_error_adjustment_config)
ssp_err_t(*	versionGet)(ssp_version_t *version)

Detailed Description

RTC driver structure. General RTC functions implemented at the HAL layer follow this API.

Field Documentation

◆ calendarAlarmGet

ssp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)

Get the calendar alarm time.

Implemented as

- R_RTC_CalendarAlarmGet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_alarm	Pointer to an alarm structure to fill up with the alarm time

◆ calendarAlarmSet

ssp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)

Set the calendar alarm time.

Implemented as

- R_RTC_CalendarAlarmSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_alarm	Pointer to an alarm structure that contains the alarm time to set
[in]	irq_enable_flag	Enable the ALARM irq if set

◆ calendarCounterStart

ssp_err_t(* rtc_api_t::calendarCounterStart) (rtc_ctrl_t *const p_ctrl)

Start the calendar counter.

Implemented as

- R_RTC_CalendarCounterStart()

Parameters

[in]	p_ctrl	Pointer to RTC device handle

◆ calendarCounterStop

ssp_err_t(* rtc_api_t::calendarCounterStop) (rtc_ctrl_t *const p_ctrl)

Stop the calendar counter.

Implemented as

- R_RTC_CalendarCounterStop()

Parameters

[in]	p_ctrl	Pointer to RTC device handle

◆ calendarTimeGet

ssp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)

Get the calendar time.

Implemented as

- R_RTC_CalendarTimeGet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_time	Pointer to a time structure that contains the time to get

◆ calendarTimeSet

ssp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)

Set the calendar time.

Implemented as

- R_RTC_CalendarTimeSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_time	Pointer to a time structure that contains the time to set
[in]	clock_start	Flag that starts the clock right after it is set

◆ close

ssp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)

Close the RTC driver.

Implemented as

- R_RTC_Close()

Parameters

[in]	p_ctrl	Pointer to RTC device handle.

◆ configure

ssp_err_t(* rtc_api_t::configure) (rtc_ctrl_t *const p_ctrl, void *const p_extend)

Configure the RTC driver.

Implemented as

- R_RTC_Configure()

Parameters

[in]	p_ctrl	Pointer to RTC device handle.
[in]	p_extend	Currently not implemented, pass NULL.

◆ errorAdjustmentModeSet

ssp_err_t(* rtc_api_t::errorAdjustmentModeSet) (rtc_ctrl_t *p_ctrl, rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode)

Set time error adjustment mode.

Implemented as

- R_RTC_ErrorAdjustmentModeSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_error_adjustment_mode	Pointer to error adjustment mode configuration structure

◆ errorAdjustmentSet

ssp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *p_ctrl, rtc_error_adjustment_cfg_t *p_error_adjustment_config)

Set time error adjustment.

Implemented as

- R_RTC_ErrorAdjustmentSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_error_adjustment_config	Pointer to error adjustment structure

◆ infoGet

ssp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)

Return the currently configure clock source for the RTC

Implemented as

- R_RTC_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_rtc_info	Pointer to RTC information structure

◆ irqDisable

ssp_err_t(* rtc_api_t::irqDisable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)

Disable the alarm irq.

Implemented as

- R_RTC_IrqDisable()

Parameters

[in]	p_ctrl	Pointer to RTC device handle

◆ irqEnable

ssp_err_t(* rtc_api_t::irqEnable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)

Enable the alarm irq.

Implemented as

- R_RTC_IrqEnable()

Parameters

[in]	p_ctrl	Pointer to RTC device handle

◆ open

ssp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)

Open the RTC driver.

Implemented as

- R_RTC_Open()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_cfg	Pointer to the configuration structure

◆ periodicIrqRateSet

ssp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)

Set the periodic irq rate

Implemented as

- R_RTC_PeriodicIrqRateSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	rate	Rate of periodic interrupts

◆ versionGet

ssp_err_t(* rtc_api_t::versionGet) (ssp_version_t *version)
Gets version and stores it in provided pointer p_version. **Implemented as** ◦ R_RTC_VersionGet() **Parameters** [out] p_version Code and API version used

---

The documentation for this struct was generated from the following file:

- r_rtc_api.h

rtc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » RTC Interface

#include <r_rtc_api.h>

Data Fields

rtc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
rtc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
rtc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_rtc_api.h

5.1.4.32 SD/MMC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for accessing SD, eMMC, and SDIO devices. More...

Data Structures

struct	sdmmc_hw_t
struct	sdmmc_info_t
struct	sdmmc_callback_args_t
struct	sdmmc_cfg_t
struct	sdmmc_api_t
struct	sdmmc_instance_t

Typedefs

typedef void	sdmmc_ctrl_t

Enumerations

enum	sdmmc_ready_status_t { SDMMC_STATUS_CARD_NOT_READY = 0x00, SDMMC_STATUS_CARD_READY }
enum	sdmmc_card_type_t { SDMMC_CARD_TYPE_MMC, SDMMC_CARD_TYPE_SD }
enum	sdmmc_media_type_t { SDMMC_MEDIA_TYPE_EMBEDDED, SDMMC_MEDIA_TYPE_CARD }
enum	sdmmc_bus_width_t { SDMMC_BUS_WIDTH_1_BIT = 1, SDMMC_BUS_WIDTH_4_BITS = 4, SDMMC_BUS_WIDTH_8_BITS = 8 }
enum	sdmmc_io_transfer_mode_t { SDMMC_IO_MODE_TRANSFER_BYTE = 0, SDMMC_IO_MODE_TRANSFER_BLOCK }
enum	sdmmc_io_address_mode_t { SDMMC_IO_ADDRESS_MODE_FIXED = 0, SDMMC_IO_ADDRESS_MODE_INCREMENT }

enum	sdmmc_io_write_mode_t { SDMMC_IO_WRITE_MODE_NO_READ = 0, SDMMC_IO_WRITE_READ_AFTER_WRITE }
enum	sdmmc_event_t { SDMMC_EVENT_CARD_REMOVED = 0x01, SDMMC_EVENT_CARD_INSERTED = 0x02, SDMMC_EVENT_ACCESS = 0x04, SDMMC_EVENT_SDIO = 0x08, SDMMC_EVENT_TRANSFER_COMPLETE = 0x10, SDMMC_EVENT_TRANSFER_ERROR = 0x20, SDMMC_EVENT_NONE = 0x00 }

Detailed Description

Interface for accessing SD, eMMC, and SDIO devices.

Summary

The r_sdmmc interface provides standard SD and eMMC media functionality. A complete file system can be implemented with FileX, sf_el_fx, sf_block_media_sdmmc and r_sdmmc modules. This driver also supports SDIO. The SD/MMC interface is implemented by:

- SDMMC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SD/MMC description: SD/MMC Driver and SDIO Driver

Typedef Documentation

◆ sdmmc_ctrl_t

typedef void sdmmc_ctrl_t

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls.

Implemented as

- sdmmc_instance_ctrl_t

Enumeration Type Documentation

◆ sdmmc_bus_width_t

enum sdmmc_bus_width_t	
SD/MMC data bus is 1, 4 or 8 bits wide.	
Enumerator	
SDMMC_BUS_WIDTH_1_BIT	Data bus is 1 bit wide.
SDMMC_BUS_WIDTH_4_BITS	Data bus is 4 bits wide.
SDMMC_BUS_WIDTH_8_BITS	Data bus is 8 bits wide.

◆ sdmmc_card_type_t

enum sdmmc_card_type_t	
SD/MMC media uses SD protocol or MMC protocol.	
Enumerator	
SDMMC_CARD_TYPE_MMC	The media is an eMMC device.
SDMMC_CARD_TYPE_SD	The media is an SD card.

◆ sdmmc_event_t

enum sdmmc_event_t	
Events that can trigger a callback function	
Enumerator	
SDMMC_EVENT_CARD_REMOVED	Card removed event.
SDMMC_EVENT_CARD_INSERTED	Card inserted event.
SDMMC_EVENT_ACCESS	Access event.
SDMMC_EVENT_SDIO	IO event.
SDMMC_EVENT_TRANSFER_COMPLETE	Read or write complete.
SDMMC_EVENT_TRANSFER_ERROR	Read or write failed.
SDMMC_EVENT_NONE	No event.

◆ sdmmc_io_address_mode_t

enum sdmmc_io_address_mode_t	
SDIO address mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_ADDRESS_MODE_FIXED	Write all data to the same address.
SDMMC_IO_ADDRESS_MODE_INCREMENT	Increment destination address after each write.

◆ sdmmc_io_transfer_mode_t

enum sdmmc_io_transfer_mode_t	
SDIO transfer mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_MODE_TRANSFER_BYTE	SDIO byte transfer mode.
SDMMC_IO_MODE_TRANSFER_BLOCK	SDIO block transfer mode.

◆ sdmmc_io_write_mode_t

enum sdmmc_io_write_mode_t	
Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register.	
Enumerator	
SDMMC_IO_WRITE_MODE_NO_READ	Write only (do not read back)
SDMMC_IO_WRITE_READ_AFTER_WRITE	Read back the register after write.

◆ sdmmc_media_type_t

enum sdmmc_media_type_t	
SD/MMC media is embedded or it can be inserted and removed.	
Enumerator	
SDMMC_MEDIA_TYPE_EMBEDDED	The media is an embedded card, or eMMC device.
SDMMC_MEDIA_TYPE_CARD	The media is an pluggable card.

◆ sdmmc_ready_status_t

enum sdmmc_ready_status_t	
SD/MMC status	
Enumerator	
SDMMC_STATUS_CARD_NOT_READY	SD card or eMMC device has not been initialized.
SDMMC_STATUS_CARD_READY	SD card or eMMC device has been initialized and is ready to access.

sdmmc_hw_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

uint8_t	channel Channel of SD/MMC host interface.
sdmmc_media_type_t	media_type Embedded or pluggable card.
sdmmc_bus_width_t	bus_width Device bus width is 1, 4 or 8 bits wide.

Detailed Description

Channel, media type, bus width defined by the hardware.

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

sdmmc_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

sdmmc_card_type_t	card_type SD or eMMC.
bool	ready
bool	hc true = Card is High Capacity card
bool	sdio true = SDIO present
bool	write_protected true = Card is write protected
bool	transfer_in_progress true = Card is busy
uint8_t	csd_version CSD version.

uint8_t	device_type Speed and data rate (eMMC)
sdmmc_bus_width_t	bus_width Current media bus width.
uint8_t	hs_timing High Speed status.
uint32_t	sdhi_rca Relative Card Address.
uint32_t	max_clock_rate Maximum clock rate for media card.
uint32_t	clock_rate Current clock rate.
uint32_t	sector_size Sector size.
uint32_t	sector_count Sector count.
uint32_t	erase_sector_count Minimum erasable unit (in 512 byte sectors)

Detailed Description

Status and other information obtained from the media device.

Field Documentation

◆ ready

bool sdmmc_info_t::ready
False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected). True otherwise. If ready is false, the driver must be closed, then reopened with a card inserted.

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

sdmmc_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

sdmmc_event_t	event The event can be used to identify what caused the callback.
void const *	p_context Placeholder for user data.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

sdmmc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

sdmmc_hw_t	hw Channel, media type, bus width defined by the hardware.
transfer_instance_t const *	p_lower_lvl_transfer Transfer instance used to move data with DMA or DTC.
void(*	p_callback)(sdmmc_callback_args_t *p_args) Pointer to callback function.
void const *	p_context User defined context passed into callback function.
void const *	p_extend SD/MMC hardware dependent configuration.
uint8_t	access_ipl Access interrupt priority.
uint8_t	sdio_ipl SDIO interrupt priority.
uint8_t	card_ipl Card interrupt priority.
uint8_t	dma_req_ipl DMA request interrupt priority.

Detailed Description

SD/MMC Configuration

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

sdmmc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

ssp_err_t(*	open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
ssp_err_t(*	close)(sdmmc_ctrl_t *const p_ctrl)
ssp_err_t(*	read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t(*	write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t(*	control)(sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)
ssp_err_t(*	readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
ssp_err_t(*	writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
ssp_err_t(*	readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
ssp_err_t(*	writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
ssp_err_t(*	ioIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	infoGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)

ssp_err_t(*	erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Detailed Description

SD/MMC functions implemented at the HAL layer API.

Field Documentation

◆ close

ssp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)

Close open SD/MMC device.

Implemented as

R_SDMMC_Close()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.

◆ control

ssp_err_t(* sdmmc_api_t::control) (sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)

The Control function sends control commands to and receives info from the SD/MMC port.

Implemented as

R_SDMMC_Control()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	command	Command to execute. The list of supported commands is below.
[in,out]	p_data	Pointer to data in or out. For each command, this data should be cast as follows: • SSP_COMMAND_GET_SECTOR_COUNT : [out] (uint32_t *) p_data • SSP_COMMAND_GET_SECTOR_SIZE : [out] (uint32_t *) p_data • SSP_COMMAND_GET_WRITE_PROTECTED : [out] (bool *) p_data • SSP_COMMAND_SET_BLOCK_SIZE : [in] (uint32_t *) p_data

◆ erase

ssp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

Implemented as

R_SDMMC_Erase

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	start_sector	First sector to erase. Must be a multiple of sdmmc_info_t::erase_sector_count.
[in]	sector_count	Number of sectors to erase. Must be a multiple of sdmmc_info_t::erase_sector_count. All sectors must be in the range of sdmmc_info_t::sector_count.

◆ infoGet

ssp_err_t(* sdmmc_api_t::infoGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)

Get SD/MMC device info.

Implemented as

R_SDMMC_InfoGet()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_info	Pointer to return device information to.

◆ IoIntEnable

ssp_err_t(* sdmmc_api_t::IoIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDMMC_IoIntEnable

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	enable	Interrupt enable = true, interrupt disable = false.

◆ open

ssp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

Open an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

Implemented as

R_SDMMC_Open()

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[in]	p_cfg	Pointer to SD/MMC instance configuration structure.

◆ read

ssp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

R_SDMMC_Read()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	start_sector	First sector address to read.
[in]	sector_count	Number of sectors to read. All sectors must be in the range of sdmmc_info_t::sector_count.

◆ readIo

ssp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDMMC_ReadIo()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_data	Pointer to location to store data byte.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.

◆ readIoExt

ssp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDMMC_ReadIoExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to read, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ versionGet

ssp_err_t(* sdmmc_api_t::versionGet) (ssp_version_t *const p_version)

Returns the version of the SD/MMC driver.

Implemented as

R_SDMMC_VersionGet()

Parameters

[out]	p_version	Pointer to return version information to.

◆ write

ssp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)

Write data to SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

R_SDMMC_Write()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	start_sector	First sector address to write to.
[in]	sector_count	Number of sectors to write. All sectors must be in the range of sdmmc_info_t::sector_count.

◆ writeIo

ssp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDMMC_WriteIo()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in,out]	p_data	Pointer to data byte to write. Read data is also provided here if read_after_write is true.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	read_after_write	Whether or not to read back the same register after writing

◆ writeIoExt

ssp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDMMC_WriteIoExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	function_number	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to write, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

sdmmc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SD/MMC Interface

#include <r_sdmmc_api.h>

Data Fields

sdmmc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.

sdmmc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
sdmmc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_sdmmc_api.h

5.1.4.33 SLCDC Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for Segment LCD controllers. More...

Data Structures

struct	slcdc_cfg_t
struct	slcdc_api_t
struct	slcdc_instance_t

Macros

#define	SLCDC_API_VERSION_MAJOR (2U)

Typedefs

typedef uint8_t	slcdc_size_t
typedef void	slcdc_ctrl_t

Enumerations

enum	slcdc_display_state_t { SLCDC_DISPLAY_STATE_CLOSED = 0, SLCDC_DISPLAY_STATE_OPENED = 1 }
enum	slcdc_bias_method_t { SLCDC_BIAS_2 = 0, SLCDC_BIAS_3,

	SLCDC_BIAS_4 }
enum	slcdc_time_slice_t { SLCDC_STATIC = 0, SLCDC_SLICE_2 = 1, SLCDC_SLICE_3 = 2, SLCDC_SLICE_4 = 3, SLCDC_SLICE_8 = 5 }
enum	slcdc_wave_form_t { SLCDC_WAVE_A = 0, SLCDC_WAVE_B }
enum	slcdc_drive_volt_gen_t { SLCDC_VOLT_EXTERNAL = 0, SLCDC_VOLT_INTERNAL, SLCDC_VOLT_CAPACITOR }
enum	slcdc_display_area_control_blink_t { SLCDC_NOT_BLINKING = 0, SLCDC_BLINKING }
enum	slcdc_display_area_t { SLCDC_DISP_A = 0, SLCDC_DISP_B, SLCDC_DISP_BLINK }
enum	slcdc_display_on_off_t { SLCDC_DISP_OFF = 0, SLCDC_DISP_ON }
enum	slcdc_display_enable_disable_t { SLCDC_DISP_DISABLE = 0, SLCDC_DISP_ENABLE }
enum	slcdc_display_clock_t { SLCDC_CLOCK_LOCO = 0x00, SLCDC_CLOCK_SOSC = 0x01, SLCDC_CLOCK_MOSC = 0x02, SLCDC_CLOCK_HOCO = 0x03 }
enum	slcdc_clk_div_t { SLCDC_CLK_DIVISOR_LOCO_4 = 1, SLCDC_CLK_DIVISOR_LOCO_8, SLCDC_CLK_DIVISOR_LOCO_16, SLCDC_CLK_DIVISOR_LOCO_32, SLCDC_CLK_DIVISOR_LOCO_64, SLCDC_CLK_DIVISOR_LOCO_128, SLCDC_CLK_DIVISOR_LOCO_256, SLCDC_CLK_DIVISOR_LOCO_512, SLCDC_CLK_DIVISOR_LOCO_1024, SLCDC_CLK_DIVISOR_HOCO_256 = 17, SLCDC_CLK_DIVISOR_HOCO_512, SLCDC_CLK_DIVISOR_HOCO_1024, SLCDC_CLK_DIVISOR_HOCO_2048, SLCDC_CLK_DIVISOR_HOCO_4096, SLCDC_CLK_DIVISOR_HOCO_8192, SLCDC_CLK_DIVISOR_HOCO_16384, SLCDC_CLK_DIVISOR_HOCO_32768, SLCDC_CLK_DIVISOR_HOCO_65536, SLCDC_CLK_DIVISOR_HOCO_131072, SLCDC_CLK_DIVISOR_HOCO_262144, SLCDC_CLK_DIVISOR_HOCO_524288 }

Detailed Description

Interface for Segment LCD controllers.

Summary

This driver uses the Segment LCD controller (SLCDC) to display data on a Segment LCD.

Implemented by: SLCDC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SLCDC Interface description: Segment LCD Driver

Macro Definition Documentation

◆ SLCDC_API_VERSION_MAJOR

#define SLCDC_API_VERSION_MAJOR (2U)
Register definitions, common services and error codes.

Typedef Documentation

◆ slcdc_ctrl_t

typedef void slcdc_ctrl_t
SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. **Implemented as** ◦ slcdc_instance_ctrl_tSLCDC control block

◆ slcdc_size_t

typedef uint8_t slcdc_size_t
Size definition for slcdc

Enumeration Type Documentation

◆ slcdc_bias_method_t

enum slcdc_bias_method_t	
LCD display bias method.	
Enumerator	
SLCDC_BIAS_2	1/2 bias method
SLCDC_BIAS_3	1/3 bias method
SLCDC_BIAS_4	1/4 bias method

◆ slcdc_clk_div_t

enum slcdc_clk_div_t	
LCD clock settings	
Enumerator	
SLCDC_CLK_DIVISOR_LOCO_4	LOCO Clock/4.
SLCDC_CLK_DIVISOR_LOCO_8	LOCO Clock/8.
SLCDC_CLK_DIVISOR_LOCO_16	LOCO Clock/16.
SLCDC_CLK_DIVISOR_LOCO_32	LOCO Clock/32.
SLCDC_CLK_DIVISOR_LOCO_64	LOCO Clock/64.
SLCDC_CLK_DIVISOR_LOCO_128	LOCO Clock/128.
SLCDC_CLK_DIVISOR_LOCO_256	LOCO Clock/256.
SLCDC_CLK_DIVISOR_LOCO_512	LOCO Clock/512.
SLCDC_CLK_DIVISOR_LOCO_1024	LOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_256	HOCO Clock/256.
SLCDC_CLK_DIVISOR_HOCO_512	HOCO Clock/512.
SLCDC_CLK_DIVISOR_HOCO_1024	HOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_2048	HOCO Clock/2048.
SLCDC_CLK_DIVISOR_HOCO_4096	HOCO Clock/4096.

SLCDC_CLK_DIVISOR_HOCO_8192	HOCO Clock/8192.
SLCDC_CLK_DIVISOR_HOCO_16384	HOCO Clock/16384.
SLCDC_CLK_DIVISOR_HOCO_32768	HOCO Clock/32768.
SLCDC_CLK_DIVISOR_HOCO_65536	HOCO Clock/65536.
SLCDC_CLK_DIVISOR_HOCO_131072	HOCO Clock/131072.
SLCDC_CLK_DIVISOR_HOCO_262144	HOCO Clock/262144.
SLCDC_CLK_DIVISOR_HOCO_524288	HOCO Clock/524288.

◆ slcdc_display_area_control_blink_t

enum slcdc_display_area_control_blink_t	
Display Data Area Control	
Enumerator	
SLCDC_NOT_BLINKING	Alternately displaying A-pattern and B-pattern area data.
SLCDC_BLINKING	Displaying an A-pattern or B-pattern area data.

◆ slcdc_display_area_t

enum slcdc_display_area_t	
Display Area data	
Enumerator	
SLCDC_DISP_A	Displaying an A-pattern area data.
SLCDC_DISP_B	Displaying an B-pattern area data.
SLCDC_DISP_BLINK	Alternatively displaying A-pattern and B-pattern area data.

◆ slcdc_display_clock_t

enum slcdc_display_clock_t	
LCD Display clock selection	
Enumerator	
SLCDC_CLOCK_LOCO	Display clock source LOCO.
SLCDC_CLOCK_SOSC	Display clock source SOSC.
SLCDC_CLOCK_MOSC	Display clock source MOSC.
SLCDC_CLOCK_HOCO	Display clock source HOCO.

◆ slcdc_display_enable_disable_t

enum slcdc_display_enable_disable_t	
LCD Display output enable	
Enumerator	
SLCDC_DISP_DISABLE	Output ground level to segment/common pin.
SLCDC_DISP_ENABLE	Output enable.

◆ slcdc_display_on_off_t

enum slcdc_display_on_off_t	
LCD Display Enable/Disable	
Enumerator	
SLCDC_DISP_OFF	Display off.
SLCDC_DISP_ON	Display on.

◆ slcdc_display_state_t

enum slcdc_display_state_t	
Display interface operation state	
Enumerator	
SLCDC_DISPLAY_STATE_CLOSED	Display closed.
SLCDC_DISPLAY_STATE_OPENED	Display opened.

◆ slcdc_drive_volt_gen_t

enum slcdc_drive_volt_gen_t	
LCD Drive Voltage Generator Select.	
Enumerator	
SLCDC_VOLT_EXTERNAL	External resistance division method.
SLCDC_VOLT_INTERNAL	Internal voltage boosting method.
SLCDC_VOLT_CAPACITOR	Capacitor split method.

◆ slcdc_time_slice_t

enum slcdc_time_slice_t	
Time slice of LCD display.	
Enumerator	
SLCDC_STATIC	Static.
SLCDC_SLICE_2	2-time slice
SLCDC_SLICE_3	3-time slice
SLCDC_SLICE_4	4-time slice
SLCDC_SLICE_8	8-time slice

◆ slcdc_wave_form_t

enum slcdc_wave_form_t	
LCD display waveform select.	
Enumerator	
SLCDC_WAVE_A	Waveform A.
SLCDC_WAVE_B	Waveform B.

slcdc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SLCDC Interface

#include <r_slcdc_api.h>

Data Fields

slcdc_display_clock_t	slcdc_clock LCD clock source (LCDSCKSEL)
slcdc_clk_div_t	slcdc_clock_setting LCD clock setting (LCDC0)
slcdc_bias_method_t	bias_method LCD display bias method select (LBAS bit).
slcdc_time_slice_t	time_slice Time slice of LCD display select (LDTY bit)
slcdc_wave_form_t	wave_form LCD display waveform select (LWAVE bit).
slcdc_drive_volt_gen_t	drive_volt_gen LCD Drive Voltage Generator Select (MDSTET bit).

Detailed Description

SLCDC configuration block

The documentation for this struct was generated from the following file:

- r_slcdc_api.h

slcdc_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SLCDC Interface

#include <r_slcdc_api.h>

Data Fields

ssp_err_t(*	open)(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
ssp_err_t(*	write)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
ssp_err_t(*	modify)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)
ssp_err_t(*	start)(slcdc_ctrl_t *const p_ctrl)
ssp_err_t(*	stop)(slcdc_ctrl_t *const p_ctrl)
ssp_err_t(*	contrastIncrease)(slcdc_ctrl_t *const p_ctrl)
ssp_err_t(*	contrastDecrease)(slcdc_ctrl_t *const p_ctrl)
ssp_err_t(*	setdisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
ssp_err_t(*	close)(slcdc_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

SLCDC functions implemented at the HAL layer will follow this API.

Field Documentation

◆ close

ssp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)

Close display device.

Implemented as

- R_SLCDC_Close()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ contrastDecrease

ssp_err_t(* slcdc_api_t::contrastDecrease) (slcdc_ctrl_t *const p_ctrl)

Decrease the display contrast. Decrease by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator

Implemented as

- R_SLCDC_ContrastDecrease()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ contrastIncrease

ssp_err_t(* slcdc_api_t::contrastIncrease) (slcdc_ctrl_t *const p_ctrl)

Increase the display contrast. Increase by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator

Implemented as

- R_SLCDC_ContrastIncrease()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ modify

ssp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)

Rewrite data in the SLCD segment. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is. Specifies the data to rewrite

Implemented as

- R_SLCDC_Modify()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	seg	Specify the segment to be written.
[in]	data_mask	Mask the data being displayed. Set 0 to the bit to be rewritten and set 1 to the other bits. Multiple bits can be rewritten. Setting value of data_mask, Bit 3 - 0xf7 Bit 2 - 0xfb Bit 1 - 0xfd Bit 0 - 0xfe
[in]	data	Specify display data to rewrite to the specified segment.

◆ open

ssp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)

Open SLCD device.

Implemented as

- R_SLCDC_Open()

Parameters

[in,out]	p_ctrl	Pointer to display interface control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to display configuration structure. All elements of this structure must be set by user.

◆ setdisplayArea

ssp_err_t(* slcdc_api_t::setdisplayArea) (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)

Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to set blink on where A-pattern and B-pattern area data will be alternately displayed.

When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic interrupt request, 1/2 second 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual: Hardware for the detailed procedure.

Implemented as

- R_SLCDC_SetdisplayArea()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	display_area	Display area to be used, A-pattern or B-pattern area.

◆ start

ssp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)

Enable display on the SLCD. Displays the specified data on the LCD. Before that data should be written to the segments.

Implemented as

- R_SLCDC_Start()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ stop

ssp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)

Disable display on the SLCD. Stops displaying data on the SLCD.

Implemented as

- R_SLCDC_Stop()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ versionGet

ssp_err_t(* slcdc_api_t::versionGet) (ssp_version_t *p_version)

Get version.

Implemented as

- R_SLCDC_VersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the version information.

◆ write

ssp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)

Write data to SLCD segment. Specifies the initial display data. Except for 8-time slice, store values in the lower 4 bits when writing to the A-pattern area, and in the upper 4 bits when writing to the B-pattern area. The display data is stored in the display data register.

Implemented as

- R_SLCDC_Write()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	start_segment	Specify the start segment number to be written.
[in]	p_data	pointer to the display data to be written to the specified segments
[in]	segment_count	Number of segments to be written

The documentation for this struct was generated from the following file:

- r_slcdc_api.h

slcdc_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SLCDC Interface

#include <r_slcdc_api.h>

Data Fields

slcdc_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
slcdc_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.

slcdc_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_slcdc_api.h

5.1.4.34 SPI Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for SPI communications. More...

Data Structures

struct	spi_callback_args_t
struct	spi_cfg_t
struct	spi_api_t
struct	spi_instance_t

Typedefs

typedef void	spi_ctrl_t

Enumerations

enum	spi_bit_width_t { SPI_BIT_WIDTH_8_BITS = (1), SPI_BIT_WIDTH_16_BITS = (2), SPI_BIT_WIDTH_32_BITS = (4) }
enum	spi_mode_t { SPI_MODE_MASTER, SPI_MODE_SLAVE }
enum	spi_clk_phase_t { SPI_CLK_PHASE_EDGE_ODD, SPI_CLK_PHASE_EDGE_EVEN }
enum	spi_clk_polarity_t { SPI_CLK_POLARITY_LOW, SPI_CLK_POLARITY_HIGH }

enum	spi_mode_fault_t { SPI_MODE_FAULT_ERROR_ENABLE, SPI_MODE_FAULT_ERROR_DISABLE }
enum	spi_bit_order_t { SPI_BIT_ORDER_MSB_FIRST, SPI_BIT_ORDER_LSB_FIRST }
enum	spi_event_t { SPI_EVENT_TRANSFER_COMPLETE = 1, SPI_EVENT_TRANSFER_ABORTED, SPI_EVENT_ERR_MODE_FAULT, SPI_EVENT_ERR_READ_OVERFLOW, SPI_EVENT_ERR_PARITY, SPI_EVENT_ERR_OVERRUN, SPI_EVENT_ERR_FRAMING, SPI_EVENT_ERR_MODE_UNDERRUN }
enum	spi_operation_t { SPI_OPERATION_DO_TX = 0x1, SPI_OPERATION_DO_RX = 0x2, SPI_OPERATION_DO_TX_RX = 0x3 }

Detailed Description

Interface for SPI communications.

Summary

The SPI Interface provides access to the SPI bus API. The Interface implements the Simple SPI on SCI HAL layer driver module.

Implemented by:

- SPI
- Simple SPI on SCI

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SPI Interface description: SPI Driver

Typedef Documentation

◆ spi_ctrl_t

typedef void spi_ctrl_t
SPI control block. Allocate an instance specific control block to pass into the SPI API calls. **Implemented as** ◦ sci_spi_instance_ctrl_t ◦ rspi_instance_ctrl_t

Enumeration Type Documentation

◆ spi_bit_order_t

enum spi_bit_order_t	
Bit-order	
Enumerator	
SPI_BIT_ORDER_MSB_FIRST	Send MSB first in transmission.
SPI_BIT_ORDER_LSB_FIRST	Send LSB first in transmission.

◆ spi_bit_width_t

enum spi_bit_width_t	
Data bit width	
Enumerator	
SPI_BIT_WIDTH_8_BITS	Data bit width is 8 bits byte.
SPI_BIT_WIDTH_16_BITS	Data bit width is 16 bits word.
SPI_BIT_WIDTH_32_BITS	Data bit width is 32 bits long word.

◆ spi_clk_phase_t

enum spi_clk_phase_t	
Clock phase	
Enumerator	
SPI_CLK_PHASE_EDGE_ODD	0: Data sampling on odd edge, data variation on even edge
SPI_CLK_PHASE_EDGE_EVEN	1: Data variation on odd edge, data sampling on even edge

◆ spi_clk_polarity_t

enum spi_clk_polarity_t	
Clock polarity	
Enumerator	
SPI_CLK_POLARITY_LOW	0: Clock polarity is low when idle
SPI_CLK_POLARITY_HIGH	1: Clock polarity is high when idle

◆ spi_event_t

enum spi_event_t	
SPI events	
Enumerator	
SPI_EVENT_TRANSFER_COMPLETE	The data transfer was completed.
SPI_EVENT_TRANSFER_ABORTED	The data transfer was aborted.
SPI_EVENT_ERR_MODE_FAULT	Mode fault error.
SPI_EVENT_ERR_READ_OVERFLOW	Read overflow error.
SPI_EVENT_ERR_PARITY	Parity error.
SPI_EVENT_ERR_OVERRUN	Overrun error.
SPI_EVENT_ERR_FRAMING	Framing error.
SPI_EVENT_ERR_MODE_UNDERRUN	Underrun error.

◆ spi_mode_fault_t

enum spi_mode_fault_t	
Mode fault error flag. This error occurs when the device is setup as a master, but the SSLA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave.	
Enumerator	
SPI_MODE_FAULT_ERROR_ENABLE	Mode fault error flag on.
SPI_MODE_FAULT_ERROR_DISABLE	Mode fault error flag off.

◆ spi_mode_t

enum spi_mode_t	
Master or slave operating mode	
Enumerator	
SPI_MODE_MASTER	Channel operates as SPI master.
SPI_MODE_SLAVE	Channel operates as SPI slave.

◆ spi_operation_t

enum spi_operation_t	
Used by control block only.	
Enumerator	
SPI_OPERATION_DO_TX	perform SPI transmission operation
SPI_OPERATION_DO_RX	perform SPI reception operation
SPI_OPERATION_DO_TX_RX	perform SPI Transmission and reception operation

spi_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SPI Interface

#include <r_spi_api.h>

Data Fields

uint32_t	channel Device channel number.
spi_event_t	event Event code.
void const *	p_context Context provided to user during callback.

Detailed Description

Common callback parameter definition

The documentation for this struct was generated from the following file:

- r_spi_api.h

spi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SPI Interface

#include <r_spi_api.h>

Data Fields

uint8_t	channel Channel number to be used.
uint8_t	rxi_ipl Receive interrupt priority.
uint8_t	txi_ipl Transmit interrupt priority.
uint8_t	tei_ipl Transmit end interrupt priority.
uint8_t	eri_ipl Error interrupt priority.
spi_mode_t	operating_mode Select master or slave operating mode.
spi_clk_phase_t	clk_phase

	Data sampling on odd or even clock edge.
spi_clk_polarity_t	clk_polarity Clock level when idle.
spi_mode_fault_t	mode_fault Mode fault error (master/slave conflict) flag.
spi_bit_order_t	bit_order Select to transmit MSB/LSB first.
uint32_t	bitrate Bits Per Second.
transfer_instance_t const *	p_transfer_tx To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.
void(*	p_callback)(spi_callback_args_t *p_args) Pointer to user callback function.
void const *	p_context User defined context passed to callback function.
void const *	p_extend Extended SPI hardware dependent configuration.

Detailed Description

SPI interface configuration

The documentation for this struct was generated from the following file:

- r_spi_api.h

spi_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SPI Interface

#include <r_spi_api.h>

Data Fields

ssp_err_t(*	open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
ssp_err_t(*	read)(spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
ssp_err_t(*	write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
ssp_err_t(*	writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
ssp_err_t(*	close)(spi_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)

Detailed Description

Shared Interface definition for SPI

Field Documentation

◆ close

ssp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

Implemented as

- R_RSPI_Close()
- R_SCI_SPI_Close()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.

◆ open

ssp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

Implemented as

- R_RSPI_Open()
- R_SCI_SPI_Open()

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to SPI configuration structure.

◆ read

ssp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Receive data from an SPI device.

Implemented as

- R_RSPI_Read()
- R_SCI_SPI_Read()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.

◆ versionGet

ssp_err_t(* spi_api_t::versionGet) (ssp_version_t *p_version)

Get the version information of the underlying driver.

Implemented as

- R_RSPI_VersionGet()
- R_SCI_SPI_VersionGet()

Parameters

[out]	p_version	pointer to memory location to return version number

◆ write

ssp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

Transmit data to an SPI device.

Implemented as

- R_RSPI_Write()
- R_SCI_SPI_Write()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ writeRead

ssp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Simultaneously transmit data to an SPI device while receiving data from a SPI device (full duplex).

Implemented as

- R_RSPI_WriteRead()
- R_SCI_SPI_WriteRead()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

The documentation for this struct was generated from the following file:

- r_spi_api.h

spi_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » SPI Interface

#include <r_spi_api.h>

Data Fields

spi_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
spi_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
spi_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_spi_api.h

5.1.4.35 Timer Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for timer functions. More...

Data Structures

struct	timer_callback_args_t
struct	timer_info_t
struct	timer_cfg_t
struct	timer_api_t
struct	timer_instance_t

Typedefs

typedef uint32_t	timer_size_t
typedef void	timer_ctrl_t

typedef timer_size_t	timer_period_t

Enumerations

enum	timer_event_t { TIMER_EVENT_EXPIRED }
enum	timer_variant_t { TIMER_VARIANT_32_BIT, TIMER_VARIANT_16_BIT }
enum	timer_status_t { TIMER_STATUS_COUNTING, TIMER_STATUS_STOPPED }
enum	timer_mode_t { TIMER_MODE_PERIODIC, TIMER_MODE_ONE_SHOT, TIMER_MODE_PWM }
enum	timer_unit_t { TIMER_UNIT_PERIOD_RAW_COUNTS, TIMER_UNIT_PERIOD_NSEC, TIMER_UNIT_PERIOD_USEC, TIMER_UNIT_PERIOD_MSEC, TIMER_UNIT_PERIOD_SEC, TIMER_UNIT_FREQUENCY_HZ, TIMER_UNIT_FREQUENCY_KHZ }
enum	timer_pwm_unit_t { TIMER_PWM_UNIT_RAW_COUNTS, TIMER_PWM_UNIT_PERCENT, TIMER_PWM_UNIT_PERCENT_X_1000 }
enum	timer_direction_t { TIMER_DIRECTION_DOWN = 0, TIMER_DIRECTION_UP = 1 }

Detailed Description

Interface for timer functions.

Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, and free-running timer mode. After each timer cycle (overflow or underflow), an interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration timer_on_<instance>_cfg_t defined in r_<instance>.h.

Implemented by:

- GPT
- AGT

See also: Input Capture Interface

Related SSP architecture topics:

- SSP Interfaces

- SSP Predefined Layers
- Using SSP Modules

Timer Interface description: Timer Driver on r_agt and Timer Driver on r_gpt

Typedef Documentation

◆ timer_ctrl_t

typedef void timer_ctrl_t
Timer control block. Allocate an instance specific control block to pass into the timer API calls. **Implemented as** ◦ gpt_instance_ctrl_t ◦ agt_instance_ctrl_t

◆ timer_period_t

typedef timer_size_t timer_period_t
DEPRECATED: Recommend using timer_size_t for period.

◆ timer_size_t

typedef uint32_t timer_size_t
Largest supported timer size. Currently up to 32-bit timers are supported. If a 16-bit timer is used, only the bottom 16 bits of any timer_size_t parameter can be used. Passing in values larger than 16 bits would result in an error.

Enumeration Type Documentation

◆ timer_direction_t

enum timer_direction_t	
Direction of timer count	
Enumerator	
TIMER_DIRECTION_DOWN	Timer count goes up.
TIMER_DIRECTION_UP	Timer count goes down.

◆ timer_event_t

enum timer_event_t	
Events that can trigger a callback function	
Enumerator	
TIMER_EVENT_EXPIRED	Requested timer delay has expired or timer has wrapped around.

◆ timer_mode_t

enum timer_mode_t	
Timer operational modes	
Enumerator	
TIMER_MODE_PERIODIC	Timer will restart after delay periods.
TIMER_MODE_ONE_SHOT	Timer will stop after delay periods.
TIMER_MODE_PWM	Timer generate PWM output.

◆ timer_pwm_unit_t

enum timer_pwm_unit_t	
Units of timer duty cycle value.	
Enumerator	
TIMER_PWM_UNIT_RAW_COUNTS	Period in clock counts.
TIMER_PWM_UNIT_PERCENT	Percent unit used for duty cycle.
TIMER_PWM_UNIT_PERCENT_X_1000	Percent multiplied by 1000 for extra resolution.

◆ timer_status_t

enum timer_status_t	
Possible status values returned by timer_api_t::infoGet.	
Enumerator	
TIMER_STATUS_COUNTING	Timer is running.
TIMER_STATUS_STOPPED	Timer is stopped.

◆ timer_unit_t

enum timer_unit_t	
Units of timer period value.	
Enumerator	
TIMER_UNIT_PERIOD_RAW_COUNTS	Period in clock counts.
TIMER_UNIT_PERIOD_NSEC	Period in nanoseconds.
TIMER_UNIT_PERIOD_USEC	Period in microseconds.
TIMER_UNIT_PERIOD_MSEC	Period in milliseconds.
TIMER_UNIT_PERIOD_SEC	Period in seconds.
TIMER_UNIT_FREQUENCY_HZ	Frequency in Hz.
TIMER_UNIT_FREQUENCY_KHZ	Frequency in kHz.

◆ timer_variant_t

enum timer_variant_t	
Timer variant types.	
Enumerator	
TIMER_VARIANT_32_BIT	32-bit timer
TIMER_VARIANT_16_BIT	16-bit timer

timer_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Timer Interface

#include <r_timer_api.h>

Data Fields

void const *	p_context
timer_event_t	event The event can be used to identify what caused the callback (overflow or error).

Detailed Description

Callback function parameter data

Field Documentation

◆ p_context

void const* timer_callback_args_t::p_context
Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t.

The documentation for this struct was generated from the following file:

- r_timer_api.h

timer_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Timer Interface

#include <r_timer_api.h>

Data Fields

timer_direction_t	count_direction Clock counting direction of the timer resource.
uint32_t	clock_frequency

	Clock frequency of the timer resource.
timer_size_t	period_counts Time in clock counts until timer will expire.
elc_event_t	elc_event ELC event associated with the count operation of the timer resource.

Detailed Description

Timer information structure to store various information for a timer resource

---

The documentation for this struct was generated from the following file:

- r_timer_api.h

timer_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Timer Interface

#include <r_timer_api.h>

Data Fields

timer_mode_t	mode Select enumerated value from timer_mode_t.
uint32_t	period
timer_unit_t	unit Units of timer_cfg_t::period.
timer_size_t	duty_cycle Duty cycle in units timer_cfg_t::duty_cycle_unit.
timer_pwm_unit_t	duty_cycle_unit

	Units of timer_cfg_t::duty_cycle.
uint8_t	channel
uint8_t	irq_ipl Timer interrupt priority.
bool	autostart
void(*	p_callback)(timer_callback_args_t *p_args)
void const *	p_context
void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

User configuration structure, used in open function

Field Documentation

◆ autostart

bool timer_cfg_t::autostart
Whether to start during Open call or not. True: Start during open. False: Don't start during open.

◆ channel

uint8_t timer_cfg_t::channel
Select a channel corresponding to the channel number of the hardware.

◆ p_callback

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)
Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void const* timer_cfg_t::p_context
Placeholder for user data. Passed to the user callback in timer_callback_args_t.

◆ period

uint32_t timer_cfg_t::period
Defines when the timer should expire. For a free running counter, set to TIMER_MAX_CLOCK with unit TIMER_UNIT_PERIOD_RAW_COUNTS

The documentation for this struct was generated from the following file:

- r_timer_api.h

timer_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Timer Interface

#include <r_timer_api.h>

Data Fields

ssp_err_t(*	open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
ssp_err_t(*	stop)(timer_ctrl_t *const p_ctrl)
ssp_err_t(*	start)(timer_ctrl_t *const p_ctrl)
ssp_err_t(*	reset)(timer_ctrl_t *const p_ctrl)
ssp_err_t(*	counterGet)(timer_ctrl_t *const p_ctrl, timer_size_t *const p_value)
ssp_err_t(*	periodSet)(timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)
ssp_err_t(*	dutyCycleSet)(timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)
ssp_err_t(*	infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

ssp_err_t(*	close)(timer_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)

Detailed Description

Timer API structure. General timer functions implemented at the HAL layer follow this API.

Field Documentation

◆ close

ssp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_GPT_Close()
- R_AGT_Close()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ counterGet

ssp_err_t(* timer_api_t::counterGet) (timer_ctrl_t *const p_ctrl, timer_size_t *const p_value)

Get current counter value and store it in provided pointer p_value.

Implemented as

- R_GPT_CounterGet()
- R_AGT_CounterGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_value	Pointer to store current counter value.

◆ dutyCycleSet

ssp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)

Sets the time until the duty cycle expires.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	duty_cycle	Time until duty cycle should expire.
[in]	duty_cycle_unit	Units of duty_cycle parameter.
[in]	pin	Which output pin to update. Enter the pin number or if pins are identified by letters, enter 0 for A, 1 for B, 2 for C, etc.

◆ infoGet

ssp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Get the time until the timer expires in clock counts and store it in provided pointer p_period_counts.

Implemented as

- R_GPT_InfoGet()
- R_AGT_InfoGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_info	Collection of information for this timer.

◆ open

ssp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_GPT_TimerOpen()
- R_AGT_TimerOpen()

Note

To reconfigure after calling this function, call timer_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ periodSet

ssp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)

Set the time until the timer expires.

Implemented as

- R_GPT_PeriodSet()
- R_AGT_PeriodSet()

Note

Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in timer_api_t::open.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	period	Time until timer should expire.
[in]	unit	Units of period parameter.

◆ reset

ssp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)

Reset the counter to the initial value.

Implemented as

- R_GPT_Reset()
- R_AGT_Reset()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ start

ssp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)

Start the counter.

Implemented as

- R_GPT_Start()
- R_AGT_Start()

Note

The counter can also be started in the timer_api_t::open function if timer_cfg_t::autostart is true.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ stop

ssp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)

Stop the counter.

Implemented as

- R_GPT_Stop()
- R_AGT_Stop()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ versionGet

ssp_err_t(* timer_api_t::versionGet) (ssp_version_t *const p_version)
Get version and store it in provided pointer p_version. **Implemented as** ◦ R_GPT_VersionGet() ◦ R_AGT_VersionGet() **Parameters** [out] p_version Code and API version used.

The documentation for this struct was generated from the following file:

- r_timer_api.h

timer_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Timer Interface

#include <r_timer_api.h>

Data Fields

timer_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
timer_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
timer_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_timer_api.h

5.1.4.36 Transfer Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for data transfer functions. More...

Data Structures

struct	transfer_properties_t
struct	transfer_info_t
struct	transfer_callback_args_t
struct	transfer_cfg_t
struct	transfer_api_t
struct	transfer_instance_t

Typedefs

typedef void	transfer_ctrl_t

Enumerations

enum	transfer_mode_t { TRANSFER_MODE_NORMAL = 0, TRANSFER_MODE_REPEAT = 1, TRANSFER_MODE_BLOCK = 2 }
enum	transfer_size_t { TRANSFER_SIZE_1_BYTE = 0, TRANSFER_SIZE_2_BYTE = 1, TRANSFER_SIZE_4_BYTE = 2 }
enum	transfer_addr_mode_t { TRANSFER_ADDR_MODE_FIXED = 0, TRANSFER_ADDR_MODE_OFFSET = 1, TRANSFER_ADDR_MODE_INCREMENTED = 2, TRANSFER_ADDR_MODE_DECREMENTED = 3 }
enum	transfer_repeat_area_t { TRANSFER_REPEAT_AREA_DESTINATION = 0, TRANSFER_REPEAT_AREA_SOURCE = 1 }
enum	transfer_chain_mode_t { TRANSFER_CHAIN_MODE_DISABLED = 0, TRANSFER_CHAIN_MODE_EACH = 2, TRANSFER_CHAIN_MODE_END = 3 }

enum	transfer_irq_t { TRANSFER_IRQ_END = 0, TRANSFER_IRQ_EACH = 1 }
enum	transfer_start_mode_t { TRANSFER_START_MODE_SINGLE = 0, TRANSFER_START_MODE_REPEAT = 1 }

Detailed Description

Interface for data transfer functions.

Summary

The transfer interface supports background data transfer (no CPU intervention).

The transfer interface can be implemented by:

- DTC
- DMAC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Transfer Interface description: Transfer Driver on r_dtc and Transfer Driver on r_dmac

Typedef Documentation

◆ transfer_ctrl_t

typedef void transfer_ctrl_t

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls.

Implemented as

- dtc_instance_ctrl_t
- dmac_instance_ctrl_t

Enumeration Type Documentation

◆ transfer_addr_mode_t

enum transfer_addr_mode_t	
Address mode specifies whether to modify (increment or decrement) pointer after each transfer.	
Enumerator	
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Address pointer changes as per the configured value of offset_byte.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

◆ transfer_chain_mode_t

enum transfer_chain_mode_t	
Chain transfer mode options. ***Note*** *Only applies for DTC.*	
Enumerator	
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

◆ transfer_irq_t

enum transfer_irq_t	
Interrupt options.	
Enumerator	
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.* Warning This will prevent chained transfers that would have happened after this one until the next activation source.

◆ transfer_mode_t

enum transfer_mode_t	
Transfer mode describes what will happen when a transfer request occurs.	
Enumerator	
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.

◆ transfer_repeat_area_t

enum transfer_repeat_area_t	
Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK, the selected pointer returns to its original value after each transfer.	
Enumerator	
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK.

◆ transfer_size_t

enum transfer_size_t	
Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t	
Enumerator	
TRANSFER_SIZE_1_BYTE	Each transfer transfers an 8-bit value.
TRANSFER_SIZE_2_BYTE	Address pointer is incremented after each transfer.
TRANSFER_SIZE_4_BYTE	Address pointer is incremented after each transfer.

◆ transfer_start_mode_t

enum transfer_start_mode_t	
Select whether to start single or repeated transfer with software start.	
Enumerator	
TRANSFER_START_MODE_SINGLE	Software start triggers single transfer.
TRANSFER_START_MODE_REPEAT	Software start transfer continues until transfer is complete.

transfer_properties_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

uint32_t	transfer_length_max Maximum number of transfers.
uint16_t	transfer_length_remaining Number of transfers remaining.
bool	in_progress Whether or not this transfer is in progress.

Detailed Description

Driver specific information.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

transfer_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

transfer_addr_mode_t	dest_addr_mode: 2
transfer_repeat_area_t	repeat_area: 1
transfer_irq_t	irq: 1
transfer_chain_mode_t	chain_mode: 2

transfer_addr_mode_t	src_addr_mode: 2
transfer_size_t	size: 2
transfer_mode_t	mode: 2
void const *volatile	p_src Source pointer.
void *volatile	p_dest Destination pointer.
volatile uint16_t	num_blocks
volatile uint16_t	length

Detailed Description

This structure specifies the properties of the transfer.

Warning

When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length.
When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t.
When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.

Note

When using DTC, consider placing instances of this structure in a protected section of memory.

Field Documentation

◆ chain_mode

transfer_chain_mode_t transfer_info_t::chain_mode
Select when the chain transfer ends.

◆ dest_addr_mode

transfer_addr_mode_t transfer_info_t::dest_addr_mode
Select what happens to destination pointer after each transfer.

◆ irq

transfer_irq_t transfer_info_t::irq
Select if interrupts should occur after each individual transfer or after the completion of all planned transfers.

◆ length

volatile uint16_t transfer_info_t::length
Length of each transfer. Range limited for TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT, see HAL driver for details.

◆ mode

transfer_mode_t transfer_info_t::mode
Select mode from transfer_mode_t.

◆ num_blocks

volatile uint16_t transfer_info_t::num_blocks
Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC an DMAC) and TRANSFER_MODE_REPEAT (DMAC only), unused in other modes.

◆ repeat_area

transfer_repeat_area_t transfer_info_t::repeat_area
Select to repeat source or destination area, unused in TRANSFER_MODE_NORMAL.

◆ size

transfer_size_t transfer_info_t::size
Select number of bytes to transfer at once. **See also** transfer_info_t::length.

◆ src_addr_mode

transfer_addr_mode_t transfer_info_t::src_addr_mode
Select what happens to source pointer after each transfer.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

transfer_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

void const *	p_context Placeholder for user data. Set in r_transfer_t::open function in transfer_cfg_t.

Detailed Description

Callback function parameter data.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

transfer_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

transfer_info_t *	p_info

elc_event_t	activation_source
bool	auto_enable
uint8_t	irq_ipl
void(*	p_callback)(transfer_callback_args_t *p_args)
void const *	p_context
void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

Driver configuration set in transfer_api_t::open. All elements except p_extend are required and must be initialized.

Field Documentation

◆ activation_source

elc_event_t transfer_cfg_t::activation_source

Select which event will trigger the transfer.

Note

Select ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 for software activation. When using DTC, these events may only be used once each. DMAC uses internal software start when either of these events are selected.

◆ auto_enable

bool transfer_cfg_t::auto_enable

Select whether the transfer should be enabled after open.

◆ irq_ipl

uint8_t transfer_cfg_t::irq_ipl

Interrupt priority level.

Warning

Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

◆ p_callback

void(* transfer_cfg_t::p_callback) (transfer_callback_args_t *p_args)
Callback for transfer end interrupt. Set to NULL for no CPU interrupt. **Warning** Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

◆ p_context

void const* transfer_cfg_t::p_context
Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

◆ p_info

transfer_info_t* transfer_cfg_t::p_info
Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

transfer_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

ssp_err_t(*	open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
ssp_err_t(*	reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)
ssp_err_t(*	enable)(transfer_ctrl_t *const p_ctrl)
ssp_err_t(*	disable)(transfer_ctrl_t *const p_ctrl)

ssp_err_t(*	start)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)
ssp_err_t(*	stop)(transfer_ctrl_t *const p_ctrl)
ssp_err_t(*	infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)
ssp_err_t(*	close)(transfer_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *const p_version)
ssp_err_t(*	blockReset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const length, transfer_size_t size, uint16_t const num_transfers)
ssp_err_t(*	Stop_ActivationRequest)(transfer_ctrl_t *const p_ctrl)

Detailed Description

Transfer functions implemented at the HAL layer will follow this API.

Field Documentation

◆ blockReset

ssp_err_t(* transfer_api_t::blockReset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const length, transfer_size_t size, uint16_t const num_transfers)

Reset source address pointer, destination address pointer, and/or length, for block transfer keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Implemented as

- R_DMAC_BlockReset()
- R_DTC_BlockReset()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	length	Transfer length in block mode.In DMAC only.
[in]	size	Transfer size in block mode. In DMAC only.
[in]	num_transfers	number of blocks in block mode. In DMAC only.

◆ close

ssp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)

Releases hardware lock. This allows a transfer to be reconfigured using transfer_api_t::open.

Implemented as

- R_DTC_Close()
- R_DMAC_Close()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ disable

ssp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)

Disable transfer. Transfers do not occur after the transfer_info_t::activation source event (or when transfer_api_t::start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source).

Note

If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.

Implemented as

- R_DMAC_Disable()
- R_DTC_Disable()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ enable

ssp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as activation source).

Implemented as

- R_DMAC_Enable()
- R_DTC_Enable()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ infoGet

ssp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)

Provides information about this transfer.

Implemented as

- R_DTC_InfoGet()
- R_DMAC_InfoGet()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[out]	p_info	Driver specific information.

◆ open

ssp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)

Initial configuration. Enables the transfer if auto_enable is true and p_src, p_dest, and length are valid. Transfers can also be enabled using transfer_api_t::enable or transfer_api_t::reset.

Implemented as

- R_DTC_Open()
- R_DMAC_Open()

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ reset

ssp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Implemented as

- R_DTC_Reset()
- R_DMAC_Reset()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	num_transfers	Transfer length in normal mode or number of blocks in block mode. In DMAC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC.

◆ start

ssp_err_t(* transfer_api_t::start) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

Start transfer in software.

Warning

Only works if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source.

Note

DTC only supports TRANSFER_START_MODE_SINGLE. DTC does not support TRANSFER_START_MODE_REPEAT.

Implemented as

- R_DMAC_Start()
- R_DTC_Start()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	mode	Select mode from transfer_start_mode_t.

◆ stop

ssp_err_t(* transfer_api_t::stop) (transfer_ctrl_t *const p_ctrl)

Stop transfer in software. The transfer will stop after completion of the current transfer.

Note

Not supported for DTC.
Only applies for transfers started with TRANSFER_START_MODE_REPEAT.

Warning

Only works if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source.

Implemented as

- R_DMAC_Stop()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ Stop_ActivationRequest

ssp_err_t(* transfer_api_t::Stop_ActivationRequest) (transfer_ctrl_t *const p_ctrl)

Clears the DMA activation request with a DMA dummy transfer as per flowchart in the hardware manual. Implements transfer_api_t::Stop_ActivationRequest.

Note

This function to be used only in scenario when a DMA activation request source might occur in the next request after a DMA transfer completes. If this happens, the DMA transfer starts and the DMA activation request is held in DMAC.

Implemented as

- R_DMAC_Stop_ActivationRequest()
- R_DTC_Stop_ActivationRequest()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ versionGet

ssp_err_t(* transfer_api_t::versionGet) (ssp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

Implemented as

- R_DTC_VersionGet()
- R_DMAC_VersionGet()

Parameters

[out]	p_version	Code and API version used.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

transfer_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » Transfer Interface

#include <r_transfer_api.h>

Data Fields

transfer_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
transfer_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
transfer_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_transfer_api.h

5.1.4.37 UART Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for UART communications. More...

Data Structures

struct	uart_info_t
struct	uart_callback_args_t
struct	uart_cfg_t
struct	uart_api_t
struct	uart_instance_t

Typedefs

typedef void	uart_ctrl_t

Enumerations

enum	uart_event_t { UART_EVENT_RX_COMPLETE = (1UL << 0), UART_EVENT_TX_COMPLETE = (1UL << 1), UART_EVENT_ERR_PARITY = (1UL << 2), UART_EVENT_ERR_FRAMING = (1UL << 3), UART_EVENT_BREAK_DETECT = (1UL << 4), UART_EVENT_ERR_OVERFLOW = (1UL << 5), UART_EVENT_ERR_RXBUF_OVERFLOW = (1UL << 6), UART_EVENT_RX_CHAR = (1UL << 7), UART_EVENT_TX_DATA_EMPTY = (1UL << 8) }
enum	uart_data_bits_t { UART_DATA_BITS_8, UART_DATA_BITS_7, UART_DATA_BITS_9 }
enum	uart_parity_t { UART_PARITY_OFF = 0U, UART_PARITY_EVEN = 2U, UART_PARITY_ODD = 3U }
enum	uart_stop_bits_t { UART_STOP_BITS_1 = 0U, UART_STOP_BITS_2 = 1U }
enum	uart_dir_t { UART_DIR_RX_TX = 0U, UART_DIR_RX = 1U, UART_DIR_TX = 2U }
enum	uart_mode_t { UART_MODE_RS232 = 0U, UART_MODE_RS485 = 1U }
enum	uart_rs485_type_t { UART_RS485_HD = 0U, UART_RS485_FD = 1U }

Detailed Description

Interface for UART communications.

Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Generic UART parameter setting
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)
- Circular buffer support
- Runtime Transmit/Receive circular buffer flushing

Implemented by:

- UART on SCI

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

UART Interface description: UART Driver

Typedef Documentation

◆ uart_ctrl_t

typedef void uart_ctrl_t
UART control block. Allocate an instance specific control block to pass into the UART API calls. **Implemented as** ◦ sci_uart_instance_ctrl_t

Enumeration Type Documentation

◆ uart_data_bits_t

enum uart_data_bits_t	
UART Data bit length definition	
Enumerator	
UART_DATA_BITS_8	Data bits 8-bit.
UART_DATA_BITS_7	Data bits 7-bit.
UART_DATA_BITS_9	Data bits 9-bit.

◆ uart_dir_t

enum uart_dir_t	
UART transaction definition	
Enumerator	
UART_DIR_RX_TX	Both RX and TX.
UART_DIR_RX	Only RX.
UART_DIR_TX	Only TX.

◆ uart_event_t

enum uart_event_t	
UART Event codes	
Enumerator	
UART_EVENT_RX_COMPLETE	Receive complete event.
UART_EVENT_TX_COMPLETE	Transmit complete event.
UART_EVENT_ERR_PARITY	Parity error event.
UART_EVENT_ERR_FRAMING	Mode fault error event.
UART_EVENT_BREAK_DETECT	Break detect error event.
UART_EVENT_ERR_OVERFLOW	FIFO Overflow error event.
UART_EVENT_ERR_RXBUF_OVERFLOW	DEPRECATED: Receive buffer overflow error event.
UART_EVENT_RX_CHAR	Character received.
UART_EVENT_TX_DATA_EMPTY	Last byte is transmitting, ready for more data.

◆ uart_mode_t

enum uart_mode_t	
UART communication mode definition	
Enumerator	
UART_MODE_RS232	Enables RS232 communication mode.
UART_MODE_RS485	Enables RS485 communication mode.

◆ uart_parity_t

enum uart_parity_t	
UART Parity definition	
Enumerator	
UART_PARITY_OFF	No parity.
UART_PARITY_EVEN	Even parity.
UART_PARITY_ODD	Odd parity.

◆ uart_rs485_type_t

enum uart_rs485_type_t	
UART RS485 communication channel type definition	
Enumerator	
UART_RS485_HD	Uses RS485 half duplex communication channel.
UART_RS485_FD	Uses RS485 full duplex communication channel.

◆ uart_stop_bits_t

enum uart_stop_bits_t	
UART Stop bits definition	
Enumerator	
UART_STOP_BITS_1	Stop bit 1-bit.
UART_STOP_BITS_2	Stop bits 2-bit.

uart_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » UART Interface

#include <r_uart_api.h>

Data Fields

uint32_t	write_bytes_max
uint32_t	read_bytes_max

Detailed Description

UART driver specific information

Field Documentation

◆ read_bytes_max

uint32_t uart_info_t::read_bytes_max

Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL.

◆ write_bytes_max

uint32_t uart_info_t::write_bytes_max

Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL.

The documentation for this struct was generated from the following file:

- r_uart_api.h

uart_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » UART Interface

#include <r_uart_api.h>

Data Fields

uint32_t	channel Device channel number.
uart_event_t	event Event code.

uint32_t	data
void const *	p_context Context provided to user during callback.

Detailed Description

UART Callback parameter definition

Field Documentation

◆ data

uint32_t uart_callback_args_t::data
Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.

The documentation for this struct was generated from the following file:

- r_uart_api.h

uart_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » UART Interface

#include <r_uart_api.h>

Data Fields

uint8_t	channel Select a channel corresponding to the channel number of the hardware.
uint32_t	baud_rate Baud rate, i.e. 9600, 19200, 115200.
uart_data_bits_t	data_bits

	Data bit length (8 or 7 or 9)
uart_parity_t	parity Parity type (none or odd or even)
uart_stop_bits_t	stop_bits Stop bit length (1 or 2)
bool	ctsrts_en CTS/RTS hardware flow control enable.
uint8_t	rxi_ipl Receive interrupt priority.
uint8_t	txi_ipl Transmit interrupt priority.
uint8_t	tei_ipl Transmit end interrupt priority.
uint8_t	eri_ipl Error interrupt priority.
transfer_instance_t const *	p_transfer_rx
transfer_instance_t const *	p_transfer_tx
void(*	p_callback)(uart_callback_args_t *p_args) Pointer to callback function.
void const *	p_context User defined context passed into callback function.
void const *	p_extend UART hardware dependent configuration.

Detailed Description

UART Configuration

Field Documentation

◆ p_transfer_rx

transfer_instance_t const* uart_cfg_t::p_transfer_rx
Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

◆ p_transfer_tx

transfer_instance_t const* uart_cfg_t::p_transfer_tx
Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

The documentation for this struct was generated from the following file:

- r_uart_api.h

uart_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » UART Interface

#include <r_uart_api.h>

Data Fields

ssp_err_t(*	open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
ssp_err_t(*	read)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)
ssp_err_t(*	write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
ssp_err_t(*	baudSet)(uart_ctrl_t *const p_ctrl, uint32_t const baudrate)
ssp_err_t(*	infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

ssp_err_t(*	close)(uart_ctrl_t *const p_ctrl)
ssp_err_t(*	versionGet)(ssp_version_t *p_version)
ssp_err_t(*	communicationAbort)(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Detailed Description

Shared Interface definition for UART

Field Documentation

◆ baudSet

ssp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, uint32_t const baudrate)

Change baud rate.

Warning

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

Implemented as

- R_SCI_UartBaudSet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	baudrate	Baud rate in bps.

◆ close

ssp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)

Close UART device.

Implemented as

- R_SCI_UartClose()

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ communicationAbort

ssp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Abort ongoing transfer.

Implemented as

- R_SCI_UartAbort()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	communication_to_abort	Type of abort request.

◆ infoGet

ssp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

Get the driver specific information.

Implemented as

- R_SCI_UartInfoGet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	baudrate	Baud rate in bps.

◆ open

ssp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

Open UART device.

Implemented as

- R_SCI_UartOpen()

Parameters

[in,out]	p_ctrl	Pointer to the UART control block Must be declared by user. Value set here.
[in]	uart_cfg_t	Pointer to UART configuration structure. All elements of this structure must be set by user.

◆ read

ssp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)

Read from UART device. If a transfer instance is used for reception, the received bytes are stored directly in the read input buffer. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet().

Implemented as

- R_SCI_UartRead()

Parameters

[in]	p_ctrl	Pointer to the UART control block for the channel.
[in]	p_dest	Destination address to read data from.
[in]	bytes	Read data length. Only applicable if uart_cfg_t::p_transfer_rx is not NULL. Otherwise all read bytes will be provided through the callback set in uart_cfg_t::p_callback.

◆ versionGet

ssp_err_t(* uart_api_t::versionGet) (ssp_version_t *p_version)

Get version.

Implemented as

- R_SCI_UartVersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the version information.

◆ write

ssp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet().

Implemented as

- R_SCI_UartWrite()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_src	Source address to write data to.
[in]	bytes	Write data length.

The documentation for this struct was generated from the following file:

- r_uart_api.h

uart_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » UART Interface

#include <r_uart_api.h>

Data Fields

uart_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
uart_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
uart_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_uart_api.h

5.1.4.38 WDT Interface

Renesas Synergy Software Package Reference » HAL Interfaces

Interface for watch dog timer functions. More...

Data Structures

struct	wdt_callback_args_t
struct	wdt_timeout_values_t
struct	wdt_cfg_t
struct	wdt_api_t
struct	wdt_instance_t

Typedefs

typedef void	wdt_ctrl_t

Enumerations

enum	wdt_timeout_t { WDT_TIMEOUT_128 = 0, WDT_TIMEOUT_512, WDT_TIMEOUT_1024, WDT_TIMEOUT_2048, WDT_TIMEOUT_4096, WDT_TIMEOUT_8192, WDT_TIMEOUT_16384 }
enum	wdt_clock_division_t { WDT_CLOCK_DIVISION_1 = 0x0000u, WDT_CLOCK_DIVISION_4 = 0x0010u, WDT_CLOCK_DIVISION_16 = 0x0020u, WDT_CLOCK_DIVISION_32 = 0x0030u, WDT_CLOCK_DIVISION_64 = 0x0040u, WDT_CLOCK_DIVISION_128 = 0x00F0u, WDT_CLOCK_DIVISION_256 = 0x0050u, WDT_CLOCK_DIVISION_512 = 0x0060u, WDT_CLOCK_DIVISION_2048 = 0x0070u, WDT_CLOCK_DIVISION_8192 = 0x0080u }
enum	wdt_window_start_t { WDT_WINDOW_START_25 = 0x0000u, WDT_WINDOW_START_50 = 0x1000u, WDT_WINDOW_START_75 = 0x2000u, WDT_WINDOW_START_100 = 0x3000u }
enum	wdt_window_end_t { WDT_WINDOW_END_75 = 0x0000u, WDT_WINDOW_END_50 = 0x0100u, WDT_WINDOW_END_25 = 0x0200u, WDT_WINDOW_END_0 = 0x0300u }
enum	wdt_reset_control_t { WDT_RESET_CONTROL_NMI = 0x00u, WDT_RESET_CONTROL_RESET = 0x80u }
enum	wdt_stop_control_t { WDT_STOP_CONTROL_DISABLE = 0x00u, WDT_STOP_CONTROL_ENABLE = 0x80u }
enum	wdt_status_t { WDT_STATUS_NO_ERROR = 0x00u, WDT_STATUS_UNDERFLOW_ERROR = 0x01u, WDT_STATUS_REFRESH_ERROR = 0x02u, WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR = 0x03u }
enum	wdt_start_mode_t { WDT_START_MODE_REGISTER = 0, WDT_START_MODE_AUTO, WDT_START_MODE_DISABLED }

Detailed Description

Interface for watch dog timer functions.

Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

See Also WDT Interface and Thread Monitor Framework Interface

The watchdog timer interface can be implemented by:

- WDT
- IWDT

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

WDT Interface description: Watchdog Driver

Typedef Documentation

◆ wdt_ctrl_t

typedef void wdt_ctrl_t
WDT control block. Allocate an instance specific control block to pass into the WDT API calls. **Implemented as** ◦ wdt_instance_ctrl_t ◦ iwdt_instance_ctrl_t

Enumeration Type Documentation

◆ wdt_clock_division_t

enum wdt_clock_division_t	
WDT clock division ratio.	
Enumerator	
WDT_CLOCK_DIVISION_1	CLK/1.
WDT_CLOCK_DIVISION_4	CLK/4.
WDT_CLOCK_DIVISION_16	CLK/16.
WDT_CLOCK_DIVISION_32	CLK/32.
WDT_CLOCK_DIVISION_64	CLK/64.
WDT_CLOCK_DIVISION_128	CLK/128.
WDT_CLOCK_DIVISION_256	CLK/256.
WDT_CLOCK_DIVISION_512	CLK/512.
WDT_CLOCK_DIVISION_2048	CLK/2048.
WDT_CLOCK_DIVISION_8192	CLK/8192.

◆ wdt_reset_control_t

enum wdt_reset_control_t	
WDT Counter underflow and refresh error control.	
Enumerator	
WDT_RESET_CONTROL_NMI	NMI request when counter underflows.
WDT_RESET_CONTROL_RESET	Reset request when counter underflows.

◆ wdt_start_mode_t

enum wdt_start_mode_t	
WDT start mode. Used to check the WDT is configured correctly.	
Enumerator	
WDT_START_MODE_REGISTER	WDT is to be configured using the WDT registers.
WDT_START_MODE_AUTO	WDT is to be configured using OFS0 hardware register.
WDT_START_MODE_DISABLED	WDT is disabled.

◆ wdt_status_t

enum wdt_status_t	
WDT status	
Enumerator	
WDT_STATUS_NO_ERROR	No status flags set.
WDT_STATUS_UNDERFLOW_ERROR	Underflow flag set.
WDT_STATUS_REFRESH_ERROR	Refresh error flag set. Refresh outside of permitted.
WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR	Underflow and refresh error flags set.

◆ wdt_stop_control_t

enum wdt_stop_control_t	
WDT Counter operation in sleep mode.	
Enumerator	
WDT_STOP_CONTROL_DISABLE	Count will not stop when device enters sleep mode.
WDT_STOP_CONTROL_ENABLE	Count will automatically stop when device enters sleep mode.

◆ wdt_timeout_t

enum wdt_timeout_t	
WDT time-out periods.	
Enumerator	
WDT_TIMEOUT_128	128 clock cycles
WDT_TIMEOUT_512	512 clock cycles
WDT_TIMEOUT_1024	1024 clock cycles
WDT_TIMEOUT_2048	2048 clock cycles
WDT_TIMEOUT_4096	4096 clock cycles
WDT_TIMEOUT_8192	8192 clock cycles
WDT_TIMEOUT_16384	16384 clock cycles

◆ wdt_window_end_t

enum wdt_window_end_t	
WDT refresh permitted period window end position.	
Enumerator	
WDT_WINDOW_END_75	End position = 75%.
WDT_WINDOW_END_50	End position = 50%.
WDT_WINDOW_END_25	End position = 25%.
WDT_WINDOW_END_0	End position = 0%.

◆ wdt_window_start_t

enum wdt_window_start_t	
WDT refresh permitted period window start position.	
Enumerator	
WDT_WINDOW_START_25	Start position = 25%.
WDT_WINDOW_START_50	Start position = 50%.
WDT_WINDOW_START_75	Start position = 75%.
WDT_WINDOW_START_100	Start position = 100%.

wdt_callback_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » WDT Interface

#include <r_wdt_api.h>

Data Fields

void const *	p_context Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t.

Detailed Description

Callback function parameter data

The documentation for this struct was generated from the following file:

- r_wdt_api.h

wdt_timeout_values_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » WDT Interface

#include <r_wdt_api.h>

Data Fields

uint32_t	clock_frequency_hz Frequency of watchdog clock after divider.
uint32_t	timeout_clocks Timeout period in units of watchdog clock ticks.

Detailed Description

WDT timeout data. Used to return frequency of WDT clock and timeout period

---

The documentation for this struct was generated from the following file:

- r_wdt_api.h

wdt_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » WDT Interface

#include <r_wdt_api.h>

Data Fields

wdt_start_mode_t	start_mode The expected start mode for the WDT.
bool	autostart
wdt_timeout_t	timeout Timeout period.
wdt_clock_division_t	clock_division Clock divider.
wdt_window_start_t	window_start

	Refresh permitted window start position.
wdt_window_end_t	window_end Refresh permitted window end position.
wdt_reset_control_t	reset_control Select NMI or reset generated on underflow.
wdt_stop_control_t	stop_control Select whether counter operates in sleep mode.
void(*	p_callback)(wdt_callback_args_t *p_args) Callback provided when a WDT NMI ISR occurs.
void const *	p_context
void const *	p_extend Placeholder for user extension.

Detailed Description

WDT configuration parameters.

Field Documentation

◆ autostart

bool wdt_cfg_t::autostart
When true the WDT is started as part of its configuration (register start mode). If false the WDT needs to be started manually by calling the refresh API.

◆ p_context

void const* wdt_cfg_t::p_context
Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

The documentation for this struct was generated from the following file:

- r_wdt_api.h

wdt_api_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » WDT Interface

#include <r_wdt_api.h>

Data Fields

ssp_err_t(*	cfgGet)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
ssp_err_t(*	open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
ssp_err_t(*	refresh)(wdt_ctrl_t *const p_ctrl)
ssp_err_t(*	statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
ssp_err_t(*	statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
ssp_err_t(*	counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
ssp_err_t(*	timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
ssp_err_t(*	versionGet)(ssp_version_t *const p_data)

Detailed Description

WDT functions implemented at the HAL layer will follow this API.

Field Documentation

◆ cfgGet

ssp_err_t(* wdt_api_t::cfgGet) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.

Implemented as

- R_WDT_CfgGet()
- R_IWDT_CfgGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_cfg	Pointer to pin configuration structure for reading WDT configuration.

◆ counterGet

ssp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)

Read the current WDT counter value.

Implemented as

- R_WDT_CounterGet()
- R_IWDT_CounterGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_count	Pointer to variable to return current WDT counter value.

◆ open

ssp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.

Implemented as

- R_WDT_Open()
- R_IWDT_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ refresh

ssp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)

Refresh the watchdog timer.

Implemented as

- R_WDT_Refresh()
- R_IWDT_Refresh()

Precondition

If the WDT is in auto-start mode ensure the OFS0 register is configured before using this function.

Warning

Calling this function in register-start mode before calling R_WDT_Open will start the WDT in it's default state and further changes to the configuration will not be possible.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ statusClear

ssp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

Clear the status flags of the WDT.

Implemented as

- R_WDT_StatusClear()
- R_IWDT_StatusClear()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	status	Status condition(s) to clear.

◆ statusGet

ssp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

Read the status of the WDT.

Implemented as

- R_WDT_StatusGet()
- R_IWDT_StatusGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to variable to return status information through.

◆ timeoutGet

ssp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

Read the watchdog timeout values.

Implemented as

- R_WDT_TimeoutGet()
- R_IWDT_TimeoutGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_timeout	Pointer to structure to return timeout values.

◆ versionGet

ssp_err_t(* wdt_api_t::versionGet) (ssp_version_t *const p_data)

Return the version of the IOPort driver.

Implemented as

- R_WDT_VersionGet()
- R_IWDT_VersionGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

The documentation for this struct was generated from the following file:

- r_wdt_api.h

wdt_instance_t Struct Reference

Renesas Synergy Software Package Reference » HAL Interfaces » WDT Interface

#include <r_wdt_api.h>

Data Fields

wdt_ctrl_t *	p_ctrl Pointer to the control structure for this instance.
wdt_cfg_t const *	p_cfg Pointer to the configuration structure for this instance.
wdt_api_t const *	p_api Pointer to the API structure for this instance.

Detailed Description

This structure encompasses everything that is needed to use an instance of this interface.

The documentation for this struct was generated from the following file:

- r_wdt_api.h

5.1.5 HAL Layer

Renesas Synergy Software Package Reference

Modules

	High-Speed Analog Comparator Driver for the High-Speed Analog Comparator.
	Low Power Analog Comparator Driver for the Low Power Analog Comparator.
	ADC Driver for the 14-Bit A/D Converter (ADC14) and 12-bit A/D Converter (ADC12).
	AGT Driver for the Asynchronous General Purpose Timer (AGT).
	AGT Input Capture Driver for the Asynchronous General-Purpose Timer (AGT) with Input Capture.
	Analog Connections Driver for internal analog connections.
	CAC Driver for the Clock Frequency Accuracy Measurement Circuit (CAC).
	CAN Driver for CAN, Controller Area Network.

	CGC Driver for the Clock Generation Circuit.
	CRC Driver for the CRC Calculator (CRC).
	CTSU v2 Driver for the Capacitive Touch Sensing Unit (CTSU).
	DAC Driver for the 12-Bit D/C Converter (DAC12).
	DAC8 Driver for the 8-Bit D/C Converter (DAC8).
	DMAC DMA Controller (DMAC).
	DOC Driver for the Data Operation Circuit (DOC).
	DTC Driver for the Data Transfer Controller (DTC).
	ELC Driver for the Event Link Controller (ELC).
	High-performance Flash Driver for the High-performance Flash Memory (S7G2 and S5D9).
	Low Power Flash Driver for the Low power Flash Memory (S3A7 and S124).

	FMI Driver for accessing Factory MCU Information (FMI).
	GLCDC Driver for the Graphics LCD Controller (GLCDC).
	GPT Driver for the General PWM Timer (GPT).
	GPT Input Capture Driver for the General PWM Timer (GPT) with Input Capture.
	ICU Driver for the Interrupt Controller Unit (ICU) External pin interrupts function.
	IOPORT Driver for the I/O Ports.
	IWDT Driver for the Independent Watchdog Timer (IWDT).
	JPEG CODEC Driver for the JPEG CODEC.
	JPEG ENCODE Driver for the JPEG CODEC.
	Key Interrupts Driver for the Key Interrupt Function.
	LPMV2 S124 Driver for Low Power Modes.

	LPMV2 S128 Driver for Low Power Modes.
	LPMV2 S1JA Driver for Low Power Modes.
	LPMV2 S3A1 Driver for Low Power Modes.
	LPMV2 S3A3 Driver for Low Power Modes.
	LPMV2 S3A6 Driver for Low Power Modes.
	LPMV2 S3A7 Driver for Low Power Modes.
	LPMV2 S5D3 Driver for Low Power Modes.
	LPMV2 S5D5 Driver for Low Power Modes.
	LPMV2 S5D9 Driver for Low Power Modes.
	LPMV2 S7G2 Driver for Low Power Modes.
	LVD Driver for Low Voltage Detection.
	Operational Amplifier (OPAMP)

	Driver for the Operational Amplifier (OPAMP).
	PDC Driver for the Parallel Data Capture Unit (PDC).
	PTP Driver for the Precision time protocol(PTP).
	PTPEDMAC DMA controller for PTP driver.
	QSPI Driver for the Quad Serial Peripheral Interface (QSPI).
	IIC Driver for the I2C Bus Interface (IIC).
	IIC Slave Driver for the I2C Bus Slave Interface (IIC Slave).
	SPI Driver for the Serial Peripheral Interface (SPI).
	RTC Driver for the Realtime Clock (RTC).
	Simple I2C on SCI Driver for the Simple IIC on SCI.
	Simple SPI on SCI Driver for the Simple SPI on SCI.
	UART on SCI Driver for the UART on SCI.

	Sigma Delta ADC (SDADC) Driver for the 24-bit Sigma Delta A/D Converter (SDADC).
	SDMMC Driver for the SD/MMC Host Interface (SDHI).
	SLCDC Driver for the Segment LCD Controller (SLCDC).
	SSI Driver for the Serial Sound Interface (SSI).
	WDT Driver for the Watchdog Timer (WDT).
	SCE Module Primitive cryptographic functions.

Detailed Description

The hardware abstraction layer provides drivers for Renesas peripherals. HAL drivers typically implement Interfaces and provide additional hardware specific APIs.

5.1.5.1 High-Speed Analog Comparator

Renesas Synergy Software Package Reference » HAL Layer

Driver for the High-Speed Analog Comparator. More...

Data Structures

struct	acmphs_instance_ctrl_t

Functions

ssp_err_t	R_ACMPHS_Open (comparator_ctrl_t *const p_api_ctrl, comparator_cfg_t const *const p_cfg)

	Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open(). More...
ssp_err_t	R_ACMPHS_InfoGet (comparator_ctrl_t *const p_api_ctrl, comparator_info_t *const p_info) Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet(). More...
ssp_err_t	R_ACMPHS_OutputEnable (comparator_ctrl_t *const p_api_ctrl) Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable(). More...
ssp_err_t	R_ACMPHS_StatusGet (comparator_ctrl_t *const p_api_ctrl, comparator_status_t *const p_status) Provides the operating status of the comparator. Implements comparator_api_t::statusGet(). More...
ssp_err_t	R_ACMPHS_Close (comparator_ctrl_t *p_api_ctrl) Stops the comparator. Implements comparator_api_t::close(). More...
ssp_err_t	R_ACMPHS_VersionGet (ssp_version_t *const p_version) Gets the API and code version. Implements comparator_api_t::versionGet(). More...

Detailed Description

Driver for the High-Speed Analog Comparator.

Summary

Extends COMPARATOR Interface.

This module implements the COMPARATOR Interface using the high-speed analog comparator.

Function Documentation

◆ R_ACMPHS_Close()

ssp_err_t R_ACMPHS_Close (comparator_ctrl_t * *p_api_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

Return values

SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

Mark driver as closed

Disable interrupts.

Stop the comparator and disable output to VCOUT.

Enter the module-stop state.

Release the hardware lock

◆ R_ACMPHS_InfoGet()

ssp_err_t R_ACMPHS_InfoGet (comparator_ctrl_t *const *p_api_ctrl*, comparator_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

Return values

SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

Get the base stabilization time.

Add 4 filter clocks if the filter is enabled.

◆ R_ACMPHS_Open()

ssp_err_t R_ACMPHS_Open (comparator_ctrl_t *const *p_api_ctrl*, comparator_cfg_t const *const *p_cfg*)

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

Return values

SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	An input pointer is NULL
SSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation.
SSP_ERR_IN_USE	The control block is already open or the hardware lock is taken.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Verify the control block has not already been initialized.

Configure interrupt priority. The interrupt is disabled until comparator_api_t::outputEnable() is called.

Enable clocks to the ACMPHS hardware.

Set registers controlled by this driver to their default values.

Configure the output polarity.

Configure the trigger edge.

Configure the hardware debounce filter.

Enable the comparator.

◆ R_ACMPHS_OutputEnable()

ssp_err_t R_ACMPHS_OutputEnable (comparator_ctrl_t *const *p_api_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

Return values

SSP_SUCCESS	Comparator output is enabled.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Enable the ACMPHS output.

Set the VCOUT output setting for this channel (enabled or disabled).

Enable interrupts for this channel.

◆ R_ACMPHS_StatusGet()

ssp_err_t R_ACMPHS_StatusGet (comparator_ctrl_t *const *p_api_ctrl*, comparator_status_t *const *p_status*)

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

Return values

SSP_SUCCESS	Operating status of the comparator is provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

Read the operating status of the comparator.

◆ R_ACMPHS_VersionGet()

ssp_err_t R_ACMPHS_VersionGet (ssp_version_t *const *p_version*)
Gets the API and code version. Implements comparator_api_t::versionGet().

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

acmphs_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » High-Speed Analog Comparator

#include <r_acmphs.h>

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in comparator_api_t::open.

The documentation for this struct was generated from the following file:

- r_acmphs.h

5.1.5.2 Low Power Analog Comparator

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Low Power Analog Comparator. More...

Data Structures

struct	acmplp_instance_ctrl_t

Functions

ssp_err_t	R_ACMPLP_Open (comparator_ctrl_t *const p_api_ctrl, comparator_cfg_t const *const p_cfg) Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called.

	comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open(). More...
ssp_err_t	R_ACMPLP_InfoGet (comparator_ctrl_t *const p_api_ctrl, comparator_info_t *const p_info) Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet(). More...
ssp_err_t	R_ACMPLP_OutputEnable (comparator_ctrl_t *const p_api_ctrl) Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable(). More...
ssp_err_t	R_ACMPLP_StatusGet (comparator_ctrl_t *const p_api_ctrl, comparator_status_t *const p_status) Provides the operating status of the comparator. Implements comparator_api_t::statusGet(). More...
ssp_err_t	R_ACMPLP_Close (comparator_ctrl_t *p_api_ctrl) Stops the comparator. Implements comparator_api_t::close(). More...
ssp_err_t	R_ACMPLP_VersionGet (ssp_version_t *const p_version) Gets the API and code version. Implements comparator_api_t::versionGet(). More...

Detailed Description

Driver for the Low Power Analog Comparator.

Summary

Extends COMPARATOR Interface.

This module implements the COMPARATOR Interface using the low power analog comparator.

Function Documentation

◆ R_ACMPLP_Close()

ssp_err_t R_ACMPLP_Close (comparator_ctrl_t * *p_api_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

Return values

SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

Mark driver as closed

Disable interrupts.

Stop the comparator and disable output to VCOUT.

Enter the module-stop state.

Release the hardware lock

◆ R_ACMPLP_InfoGet()

ssp_err_t R_ACMPLP_InfoGet (comparator_ctrl_t *const *p_api_ctrl*, comparator_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

Return values

SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

◆ R_ACMPLP_Open()

ssp_err_t R_ACMPLP_Open (comparator_ctrl_t *const *p_api_ctrl*, comparator_cfg_t const *const *p_cfg*)

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

Return values

SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	An input pointer is NULL
SSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation.
SSP_ERR_IN_USE	The control block is already open or the hardware lock is taken.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Verify the control block has not already been initialized.

Configure interrupt priority. The interrupt is disabled until comparator_api_t::outputEnable() is called.

Enable clocks to the ACMPLP hardware.

Set registers controlled by this channel to their default values.

Set the mode.

Configure the output polarity.

Configure the trigger edge.

Configure the hardware debounce filter.

Enable the comparator.

◆ R_ACMPLP_OutputEnable()

ssp_err_t R_ACMPLP_OutputEnable (comparator_ctrl_t *const *p_api_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

Return values

SSP_SUCCESS	Comparator output is enabled.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Set the VCOUT output setting for this channel (enabled or disabled).

Enable interrupts for this channel.

◆ R_ACMPLP_StatusGet()

ssp_err_t R_ACMPLP_StatusGet (comparator_ctrl_t *const *p_api_ctrl*, comparator_status_t *const *p_status*)

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

Return values

SSP_SUCCESS	Operating status of the comparator is provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Read the operating status of the comparator.

◆ R_ACMPLP_VersionGet()

ssp_err_t R_ACMPLP_VersionGet (ssp_version_t *const *p_version*)

Gets the API and code version. Implements comparator_api_t::versionGet().

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

acmplp_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Low Power Analog Comparator

#include <r_acmplp.h>

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in comparator_api_t::open.

The documentation for this struct was generated from the following file:

- r_acmplp.h

5.1.5.3 ADC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the 14-Bit A/D Converter (ADC14) and 12-bit A/D Converter (ADC12). More...

Data Structures

struct	adc_instance_ctrl_t

Macros

#define	ADC_CODE_VERSION_MAJOR (2U)
#define	ADC_SAMPLE_STATE_COUNT_MIN (7U)
#define	ADC_SAMPLE_STATE_HOLD_COUNT_MIN (4U)
#define	ADC_SAMPLE_STATE_HOLD_COUNT_DEFAULT (24U)
#define	ADC_MASK_CHANNEL_0 (1U<<0U)
#define	ADC_SAMPLE_HOLD_CHANNELS (0x07U)

Functions

ssp_err_t	R_ADC_Open (adc_ctrl_t *p_api_ctrl, adc_cfg_t const *const p_cfg) The Open function applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt priority is

	non-zero in BSP_IRQ_Cfg.h, the function takes a callback function pointer for notifying the user at interrupt level whenever a scan has completed. On MCUs where calibration is possible, this function will only return after calibration is completed if enabled in the user configuration. The calibration times vary depending on PCLKB and ADCLK. More...
ssp_err_t	R_ADC_SetSampleStateCount (adc_ctrl_t *p_api_ctrl, adc_sample_state_t *p_sample) Set the sample state count for individual channels. This only needs to be set for special use cases. Normally, use the default values out of Reset. More...
ssp_err_t	R_ADC_ScanConfigure (adc_ctrl_t *p_api_ctrl, adc_channel_cfg_t const *const p_channel_cfg) Configure the ADC scan parameters. Channel specific settings are set in this function. More...
ssp_err_t	R_ADC_InfoGet (adc_ctrl_t *p_api_ctrl, adc_info_t *p_adc_info) This function returns the address of the lowest number configured channel and the total number of bytes to be read in order to read the results of the configured channels and return the ELC Event name. If no channels are configured, then a length of 0 is returned. This function retrieves the temperature sensor slope. It also returns the calibration data for the sensor if available on this MCU otherwise an invalid calibration data of 0xFFFFFFFF will be returned. More...
ssp_err_t	R_ADC_ScanStart (adc_ctrl_t *p_api_ctrl) This function starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the Open() call. If the Unit was configured for hardware triggering, then this function simply allows the trigger signal (hardware or software) to get to the ADC Unit. The function is not able to control the generation of the trigger itself. If the Unit was configured for software triggering, then this function starts the software triggered scan. More...
ssp_err_t	R_ADC_ScanStop (adc_ctrl_t *p_api_ctrl) This function stops the software scan or disables the Unit from being triggered by the hardware trigger (internal or external) based on what type of trigger the unit was configured for in the Open() function. Stopping a hardware triggered scan via this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software triggered scan aborts an ongoing scan. More...

ssp_err_t	R_ADC_CheckScanDone (adc_ctrl_t *p_api_ctrl) This function returns the status of any scan process that was started. On supported MCUs, the status of the ADC calibration is returned. More...
ssp_err_t	R_ADC_Read (adc_ctrl_t *p_api_ctrl, adc_register_t const reg_id, adc_data_size_t *const p_data) This function reads conversion results from a single channel or sensor register. More...
ssp_err_t	R_ADC_Read32 (adc_ctrl_t *p_api_ctrl, adc_register_t const reg_id, uint32_t *const p_data) This function reads conversion results from a single channel or sensor register into a 32-bit result. More...
ssp_err_t	R_ADC_Close (adc_ctrl_t *p_api_ctrl) This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral. More...
ssp_err_t	R_ADC_VersionGet (ssp_version_t *const p_version) Retrieve the API version number. More...
ssp_err_t	R_ADC_Calibrate (adc_ctrl_t *const p_api_ctrl, void *const p_extend) This function initiates calibration of the ADC on supported MCUs. Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. If ADC interrupts are enabled, a notification is provided via callback when calibration is complete. Otherwise, if the ADC interrupts are disabled then no notification will be provided and the application must check calibration status using infoGet() to determine if the calibration is complete before using the ADC API.Interrupts are enabled in adc_api_t::scanStatusGet(). More...
ssp_err_t	R_ADC_OffsetSet (adc_ctrl_t *const p_api_ctrl, adc_register_t const reg_id, int32_t offset)

Detailed Description

Driver for the 14-Bit A/D Converter (ADC14) and 12-bit A/D Converter (ADC12).

This module supports the ADC14 and ADC12 peripherals. It implements the following interfaces:

- ADC Interface

Macro Definition Documentation

◆ ADC_CODE_VERSION_MAJOR

#define ADC_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ ADC_MASK_CHANNEL_0

#define ADC_MASK_CHANNEL_0 (1U<<0U)
For ADC Scan configuration adc_channel_cfg_t::scan_mask, scan_mask_group_b, add_mask and sample_hold_mask Use bitwise OR to combine these masks for desired channels and sensors.

◆ ADC_SAMPLE_HOLD_CHANNELS

#define ADC_SAMPLE_HOLD_CHANNELS (0x07U)
Sample and hold Channel mask. Sample and hold is only available for channel 0,1,2

◆ ADC_SAMPLE_STATE_COUNT_MIN

#define ADC_SAMPLE_STATE_COUNT_MIN (7U)
Typical values that can be used to modify the sample states. The minimum sample state count value is either 6 or 7 depending on the clock ratios. It is fixed to 7 based on the fact that at the lowest ADC conversion clock supported (1 MHz) this extra state will lead to at worst a "1 microsecond" increase in conversion time. At 60 MHz the extra sample state will add 16.7 ns to the conversion time.

◆ ADC_SAMPLE_STATE_HOLD_COUNT_DEFAULT

#define ADC_SAMPLE_STATE_HOLD_COUNT_DEFAULT (24U)
Default sample and hold states

◆ ADC_SAMPLE_STATE_HOLD_COUNT_MIN

#define ADC_SAMPLE_STATE_HOLD_COUNT_MIN (4U)
Typical values that can be used for the sample and hold counts for the channels 0-2 Minimum sample and hold states

Function Documentation

◆ R_ADC_Calibrate()

ssp_err_t R_ADC_Calibrate (adc_ctrl_t *const *p_api_ctrl*, void *const *p_extend*)

This function initiates calibration of the ADC on supported MCUs. Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. If ADC interrupts are enabled, a notification is provided via callback when calibration is complete. Otherwise, if the ADC interrupts are disabled then no notification will be provided and the application must check calibration status using infoGet() to determine if the calibration is complete before using the ADC API.Interrupts are enabled in adc_api_t::scanStatusGet().

Parameters

[in]	p_api_ctrl	Pointer to control handle structure
[in]	p_extend	Unused argument. Pass NULL.

Return values

SSP_SUCCESS	Calibration successfully initiated.
SSP_ERR_INVALID_HW_CONDITION	Hardware is in invalid state to perform calibration due to ongoing scan or scan trigger is enabled.
SSP_ERR_UNSUPPORTED	Calibration not supported on this MCU.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.

Perform parameter checking

ADC Calibration can only happen if there is no ongoing scan and if the scan trigger is disabled

Set the normal mode interrupt request to occur when calibration is complete

Initiate calibration

Return the unsupported error.

◆ R_ADC_CheckScanDone()

ssp_err_t R_ADC_CheckScanDone (adc_ctrl_t * *p_api_ctrl*)

This function returns the status of any scan process that was started. On supported MCUs, the status of the ADC calibration is returned.

Return values

SSP_SUCCESS	Successful; the scan is complete.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_IN_USE	Running scan or calibration is still in progress.

Note

If the peripheral was configured in single scan mode, then the return value of this function is an indication of the scan status. However, if the peripheral was configured in group mode, then the return value of this function could be an indication of either the group A or group B scan state. This is because the ADST bit is set when a scan is ongoing and cleared when the scan is done. This function should normally only be used when using software trigger in single scan mode.

Perform parameter checking

Ensure ADC Unit is already open

Read status of ADC calibration and return busy status if calibration is ongoing

Read the status of the ADST bit

Return the error code

◆ R_ADC_Close()

ssp_err_t R_ADC_Close (adc_ctrl_t * *p_api_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.

Perform parameter checking

Verify that the ADC is already open

Mark driver as closed

Perform hardware stop for the specific unit

Release the lock

Return the error code

◆ R_ADC_InfoGet()

ssp_err_t R_ADC_InfoGet (adc_ctrl_t * *p_api_ctrl*, adc_info_t * *p_adc_info*)

This function returns the address of the lowest number configured channel and the total number of bytes to be read in order to read the results of the configured channels and return the ELC Event name. If no channels are configured, then a length of 0 is returned. This function retrieves the temperature sensor slope. It also returns the calibration data for the sensor if available on this MCU otherwise an invalid calibration data of 0xFFFFFFFF will be returned.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::eventInfoGet

Note

: Currently this function call does not support Group Mode operation.

Verify the parameters are valid

Return an error if the parameter check failed

Get a pointer to the base register for the current unit

Retrieve the scan mask of active channels from the control structure

If at least one channel is configured, determine the highest and lowest configured channels

Determine the lowest channel that is configured

Determine the highest channel that is configured

Determine the size of data that must be read to read all the channels between and including the highest and lowest channels.

If no channels are configured, set the return length 0

Specify the peripheral name in the ELC list

Verify the return value from fmi event information

Set Temp Sensor calibration data to invalid value

If calibration register is available, retrieve it from the MCU

Provide the previously retrieved slope information

◆ R_ADC_OffsetSet()

ssp_err_t R_ADC_OffsetSet (adc_ctrl_t *const *p_api_ctrl*, adc_register_t const *reg_id*, int32_t *offset*)
adc_api_t::offsetSet is not supported on the ADC. Return the unsupported error.

◆ R_ADC_Open()

ssp_err_t R_ADC_Open (adc_ctrl_t * *p_api_ctrl*, adc_cfg_t const *const *p_cfg*)
The Open function applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt priority is non-zero in BSP_IRQ_Cfg.h, the function takes a callback function pointer for notifying the user at interrupt level whenever a scan has completed. On MCUs where calibration is possible, this function will only return after calibration is completed if enabled in the user configuration. The calibration times vary depending on PCLKB and ADCLK.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_cfg is NULL.
SSP_ERR_INVALID_ARGUMENT	Mode or element of p_cfg structure has invalid value or is illegal based on mode.
SSP_ERR_IN_USE	Calibration timed out.

Perform parameter checking

Verify this unit has not already been initialized

Set all p_ctrl fields prior to using it in any functions

Save callback function pointer

Store the Unit number into the control structure

Store the user context into the control structure

Store the mode into the control structure

Store the alignment into the control structure

Save the regular mode/Group A trigger in the internal control block

Save the context

Store the voltage reference into the control structure

Store the over_current into the control structure

pga0 setting

pga1 setting

pga2 setting

Confirm the requested unit exists on this MCU and record available channels.

Lock specified ADC channel

Retrieve temperature sensor information into control block

Set ADC and Temperature sensors to a stop state

Initialize the hardware based on the configuration

Set ADC and Temperature sensors to a stop state

Configure PGA for the supported MCU's

Invalid scan mask (initialized for later).

Mark driver as opened by initializing it to "RADC" in its ASCII equivalent for this unit.

Return the error code

◆ R_ADC_Read()

ssp_err_t R_ADC_Read (adc_ctrl_t * *p_api_ctrl*, adc_register_t const *reg_id*, adc_data_size_t *const *p_data*)

This function reads conversion results from a single channel or sensor register.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Perform parameter checking

Verify that the ADC is already open

Get pointer to appropriate base address. This is repeated here in case parameter checking is disabled.

Read the data from the requested ADC conversion register and return it

Return the error code

◆ R_ADC_Read32()

ssp_err_t R_ADC_Read32 (adc_ctrl_t * *p_api_ctrl*, adc_register_t const *reg_id*, uint32_t *const *p_data*)

This function reads conversion results from a single channel or sensor register into a 32-bit result.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Read the 16-bit result.

Left shift the result into the upper 16 bits if the unit is configured for left alignment.

◆ R_ADC_ScanConfigure()

ssp_err_t R_ADC_ScanConfigure (adc_ctrl_t * *p_api_ctrl*, adc_channel_cfg_t const *const *p_channel_cfg*)

Configure the ADC scan parameters. Channel specific settings are set in this function.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_channel_cfg is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Note

If the Group Mode Priority configuration is set to ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN, then since Group B will be scanning continuously, Group B Interrupts are disabled and the application will not receive a callback for Group B scan completion even if a callback is provided. The application will still receive a callback for Group A scan completion if a callback is provided.
If the ADC conversion clock is faster than 50 MHz, the Temperature and Voltage sensor will not be accurate across the operating temperature range, so an error is returned.

Perform parameter checking

Ensure ADC Unit is already open

Configure the hardware based on the configuration

Save the scan mask locally; this is required for the infoGet function

Return the error code

◆ R_ADC_ScanStart()

ssp_err_t R_ADC_ScanStart (adc_ctrl_t * *p_api_ctrl*)

This function starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the Open() call. If the Unit was configured for hardware triggering, then this function simply allows the trigger signal (hardware or software) to get to the ADC Unit. The function is not able to control the generation of the trigger itself. If the Unit was configured for software triggering, then this function starts the software triggered scan.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_IN_USE	Running scan is still in progress

Perform parameter checking

Ensure ADC Unit is already open

If the the normal/GroupA trigger is not set to software, then that the Unit is configured for hardware triggering

Otherwise, enable software triggering

Check to see if there is an ongoing scan else start the scan

Return the error code

◆ R_ADC_ScanStop()

ssp_err_t R_ADC_ScanStop (adc_ctrl_t * *p_api_ctrl*)

This function stops the software scan or disables the Unit from being triggered by the hardware trigger (internal or external) based on what type of trigger the unit was configured for in the Open() function. Stopping a hardware triggered scan via this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software triggered scan aborts an ongoing scan.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.

Note

Stopping a software scan results in immediate stoppage of the scan irrespective of current state of of the scan. Stopping the hardware scan results in disabling the trigger to prevent future scans from starting but does not affect the current scan.

Perform parameter checking

Ensure ADC Unit is already open

If the trigger is not software scan, then disallow hardware triggering

Otherwise, disable software triggering

Return the error code

◆ R_ADC_SetSampleStateCount()

ssp_err_t R_ADC_SetSampleStateCount (adc_ctrl_t * *p_api_ctrl*, adc_sample_state_t * *p_sample*)

Set the sample state count for individual channels. This only needs to be set for special use cases. Normally, use the default values out of Reset.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_sample is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Perform parameter checking

Ensure ADC Unit is already open

Set the sample state count for the specified register

Return the error code

◆ R_ADC_VersionGet()

ssp_err_t R_ADC_VersionGet (ssp_version_t *const *p_version*)
Retrieve the API version number. **Return values** SSP_SUCCESS — Successful return. SSP_ERR_ASSERTION — The parameter p_version is NULL. Return the version number

adc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » ADC

#include <r_adc.h>

Data Fields

uint16_t	unit ADC Unit in use.
int16_t	slope_microvolts Temperature sensor slope in microvolts/°C.
adc_mode_t	mode operational mode
adc_alignment_t	alignment alignment
uint8_t	max_resolution ADC max resolution: 8, 10, 12, or 14-bit.
uint8_t	pga_available PGA available or not on MCU.

uint8_t	tsn_ctrl_available Availability of TSN control register.
uint8_t	tsn_calib_available Availability of TSn calibration register.
uint8_t	adc_calib_available Availability of ADC calibration feature.
R_TSN_Control_Type *	p_tsn_ctrl_regs Pointer to temperature control register.
R_TSN_Calibration_Type *	p_tsn_calib_regs Pointer to temperature calibration register.
void const *	p_context Placeholder for user data.
void *	p_reg Base register for this unit.
void(*	callback)(adc_callback_args_t *p_args) User callback pointer.
adc_trigger_t	trigger Trigger defined for normal mode.
uint32_t	opened Boolean to verify that the Unit has been initialized.
uint32_t	scan_mask Scan mask used for Normal scan.

IRQn_Type	scan_end_irq Scan end IRQ number.
IRQn_Type	scan_end_b_irq Scan end group B IRQ number.
adc_voltage_reference_t	voltage_ref ADC reference voltage selection. Default is VREF.
adc_over_current_t	over_current ADC reference voltage selection. Default is Over current.
adc_pga_t	pga0 PGA0 setting.
adc_pga_t	pga1 PGA1 setting.
adc_pga_t	pga2 PGA2 setting.

Detailed Description

ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().

The documentation for this struct was generated from the following file:

- r_adc.h

5.1.5.4 AGT

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Asynchronous General Purpose Timer (AGT). More...

Data Structures

struct	agt_instance_ctrl_t
struct	timer_on_agt_cfg_t

Enumerations

enum	agt_count_source_t { AGT_CLOCK_PCLKB = 0, AGT_CLOCK_PCLKB_DIV_8 = 1, AGT_CLOCK_PCLKB_DIV_2 = 3, AGT_CLOCK_LOCO = 4, AGT_CLOCK_AGT0_UNDERFLOW = 5, AGT_CLOCK_FSUB = 6 }

Functions

ssp_err_t	R_AGT_TimerOpen (timer_ctrl_t *const p_api_ctrl, timer_cfg_t const *const p_cfg) Open the AGT channel as a timer, handles required initialization described in hardware manual. Implements timer_api_t::open. More...
ssp_err_t	R_AGT_Close (timer_ctrl_t *const p_api_ctrl)
ssp_err_t	R_AGT_CounterGet (timer_ctrl_t *const p_api_ctrl, timer_size_t *const p_value)
ssp_err_t	R_AGT_PeriodSet (timer_ctrl_t *const p_api_ctrl, timer_size_t const period, timer_unit_t const unit)
ssp_err_t	R_AGT_DutyCycleSet (timer_ctrl_t *const p_api_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const unit, uint8_t const pin)
ssp_err_t	R_AGT_Reset (timer_ctrl_t *const p_api_ctrl)
ssp_err_t	R_AGT_Start (timer_ctrl_t *const p_api_ctrl)
ssp_err_t	R_AGT_InfoGet (timer_ctrl_t *const p_api_ctrl, timer_info_t *const p_info) Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet. More...
ssp_err_t	R_AGT_Stop (timer_ctrl_t *const p_api_ctrl)
ssp_err_t	R_AGT_VersionGet (ssp_version_t *const p_version)

Detailed Description

Driver for the Asynchronous General Purpose Timer (AGT).

Summary

Extends Timer Interface.

HAL High-Level Driver for accessing and configuring AGT timer modes.

The AGT timer functions are used by the Timer to provide timer services.

Enumeration Type Documentation

◆ agt_count_source_t

enum agt_count_source_t	
Count source	
Enumerator	
AGT_CLOCK_PCLKB	Counter clock source is PCLKB when AGT_CLOCK_PCLKB, AGT_CLOCK_PCLKB_DIV_2, or AGT_CLOCK_PCLKB_DIV_8 is selected. The PCLKB divisor is selected automatically at runtime to the optimal value of PCLKB/1, PCLKB/2, or PCLKB/8. If the timer_cfg_t::unit is TIMER_UNIT_PERIOD_RAW_COUNTS, the timer_cfg_t::period should be the desired value in PCLKB counts, even if the value would exceed 16 bits. For example, if a period of 0x30000 counts is requested, a divisor of PCLKB/8 is be selected and the counter underflows after 0x6000 counts.
AGT_CLOCK_PCLKB_DIV_8	Superseded: See AGT_CLOCK_PCLKB.
AGT_CLOCK_PCLKB_DIV_2	Superseded: See AGT_CLOCK_PCLKB.
AGT_CLOCK_LOCO	Divided clock LOCO specified y bits CKS[2:0] in the AGTMR2 register.
AGT_CLOCK_AGT0_UNDERFLOW	Underflow event signal from AGT0.
AGT_CLOCK_FSUB	Divided clock fSUB specified by bits CKS[2:0] in the AGTMR2 register.

Function Documentation

◆ R_AGT_Close()

ssp_err_t R_AGT_Close (timer_ctrl_t *const *p_api_ctrl*)

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.

Return values

SSP_SUCCESS	The AGT Timer channel is successfully closed.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The AGT channel is not opened.

Cleanup the device: Stop counter, disable interrupts, and power down if no other channels are in use.

Clear the TOE (output enable) bit

Clear the TEDGSEL bit

Unlock channel

◆ R_AGT_CounterGet()

ssp_err_t R_AGT_CounterGet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t *const *p_value*)

Retrieve and store counter value in provided p_value pointer. Implements timer_api_t::counterGet.

Return values

SSP_SUCCESS	Counter value read, p_value is valid.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null
SSP_ERR_NOT_OPEN	The channel is not opened.

Read counter value

Increment retrieved counter value by one, as counter counts till zero

◆ R_AGT_DutyCycleSet()

ssp_err_t R_AGT_DutyCycleSet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t const *duty_cycle*, timer_pwm_unit_t const *unit*, uint8_t const *pin*)

Setting duty cycle is not supported by this driver. Implements timer_api_t::dutyCycleSet.

Return values

SSP_SUCCESS	Once duty cycle set successfully.
SSP_ERR_INVALID_ARGUMENT	If any of the argument is invalid
SSP_ERR_NOT_OPEN	The channel is not opened.

Set duty cycle.

◆ R_AGT_InfoGet()

ssp_err_t R_AGT_InfoGet (timer_ctrl_t *const *p_api_ctrl*, timer_info_t *const *p_info*)

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Return values

SSP_SUCCESS	Period, status, count direction, frequency value written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_period_counts parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INVALID_HW_CONDITION	Invalid hardware setting is detected.

Get and store period

Get and store clock frequency

AGT supports only counting down direction

◆ R_AGT_PeriodSet()

ssp_err_t R_AGT_PeriodSet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t const *period*, timer_unit_t const *unit*)

Sets period value provided. Implements timer_api_t::periodSet.

Return values

SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARG	One of the following is invalid: • p_period->unit: must be one of the options from timer_size_t::unit • p_period->value: must result in a period supported by the clock source specified during the Open call.
SSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- fmi_api_t::productFeatureGet
- cgc_api_t::systemClockFreqGet

Make sure period is valid, then set period

◆ R_AGT_Reset()

ssp_err_t R_AGT_Reset (timer_ctrl_t *const *p_api_ctrl*)

Resets the counter value to the period that was originally set. Implements timer_api_t::reset.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Save running status for restart, if already running.

Set AGT counter for given period value.

Restart the AGT channel if it was running.

◆ R_AGT_Start()

ssp_err_t R_AGT_Start (timer_ctrl_t *const *p_api_ctrl*)

Starts timer. Implements timer_api_t::start.

Return values

SSP_SUCCESS	Timer successfully started.
SSP_ERR_ASSERTION	The p_ctrl parameter is null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Start timer

◆ R_AGT_Stop()

ssp_err_t R_AGT_Stop (timer_ctrl_t *const *p_api_ctrl*)

Stops the AGT channel specified by the handle (control block). This API implements timer_api_t::stop. This API does not reset the channel or power it down.

Return values

SSP_SUCCESS	Timer successfully stopped.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_AGT_TimerOpen()

ssp_err_t R_AGT_TimerOpen (timer_ctrl_t *const *p_api_ctrl*, timer_cfg_t const *const *p_cfg*)

Open the AGT channel as a timer, handles required initialization described in hardware manual. Implements timer_api_t::open.

The Timer Open function configures a single AGT channel for timer mode with parameters specified in the timer Configuration structure. It also sets up the control block for use with subsequent AGT Timer APIs.

This function must be called once prior to calling any other AGT API functions. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

The AGT hardware does not support one-shot functionality natively. The one-shot feature is therefore implemented in the AGT HAL layer. For a timer configured as a one-shot timer, the timer is stopped upon the first timer expiration.

The AGT implementation of the general timer can accept an optional timer_on_agt_cfg_t extension parameter. For AGT, the extension specifies the clock to be used as timer source and the output pin configurations. If the extension parameter is not specified (NULL), the default clock PCLKB is

used and the output pins are disabled.

The clock divider is selected based on the source clock frequency and the timer period supplied by the caller.

Return values

SSP_SUCCESS	Initialization was successful and timer has started.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_cfg, p_ctrl, or the configuration channel ID exceeds AGT_MAX_CH, or the configuration mode is invalid.
SSP_ERR_IN_USE	The channel specified has already been opened.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt has not been enabled in the BSP.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- cgc_api_t::systemClockFreqGet
- cgc_api_t::clockCheck

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Check agt count source to throw an error.

Verify channel is not already used

Power on the AGT channel.

Wait for counter to stop.

Clear AGTO output

Clear TEDGSEL bit to normal output.

Set the AGT mode based on agt_mode value.

Make sure period is valid, then set period

Start the timer if requested by user

All done.

◆ R_AGT_VersionGet()

ssp_err_t R_AGT_VersionGet (ssp_version_t *const *p_version*)
Sets driver version based on compile time macros. Implements timer_api_t::versionGet. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — The parameter p_version is NULL.

agt_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » AGT

#include <r_agt.h>

Data Fields

void(*	p_callback)(timer_callback_args_t *p_args)
void const *	p_context
void *	p_reg Base register for this channel.
uint32_t	open Whether or not channel is open.
uint16_t	period Current timer period (counts)
uint8_t	channel Channel number.
IRQn_Type	irq Counter overflow IRQ number.

timer_mode_t	mode Timer mode.

Detailed Description

Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

Field Documentation

◆ p_callback

void(* agt_instance_ctrl_t::p_callback) (timer_callback_args_t *p_args)
Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.

◆ p_context

void const* agt_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in timer_callback_args_t.

The documentation for this struct was generated from the following file:

- r_agt.h

timer_on_agt_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » AGT

#include <r_agt.h>

Data Fields

agt_count_source_t	count_source AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_CLOCK_FSUB.
bool	agto_output_enabled AGTO pin is enabled for output compare (true, false)

bool	agtio_output_enabled AGTIO pin is enabled for output compare (true, false)
bool	output_inverted Output inverted (true, false)
bool	agtoa_output_enable Enable comparator A output pin (true, false)
bool	agtob_output_enable Enable comparator B output pin (true, false)

Detailed Description

Optional AGT extension data structure.

The documentation for this struct was generated from the following file:

- r_agt.h

5.1.5.5 AGT Input Capture

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Asynchronous General-Purpose Timer (AGT) with Input Capture. More...

Data Structures

struct	agt_input_capture_extend_t Extension configuration struct for AGT Input Capture. More...
struct	agt_input_capture_instance_ctrl_t

Enumerations

enum	agt_input_capture_count_source_t { AGT_INPUT_CAPTURE_CLOCK_PCLKB = 0U, AGT_INPUT_CAPTURE_CLOCK_PCLKB_DIV_8 = 1U, AGT_INPUT_CAPTURE_CLOCK_PCLKB_DIV_2 = 3U,

	AGT_INPUT_CAPTURE_CLOCK_LOCO = 4U, AGT_CLOCK_INPUT_CAPTURE_FSUB = 6U }
enum	agt_input_capture_count_edges_t { INPUT_CAPTURE_SIGNAL_SINGLE_EDGE, INPUT_CAPTURE_SIGNAL_BOTH_EDGE }
enum	agt_input_capture_signal_filter_t { AGT_INPUT_CAPTURE_SIGNAL_FILTER_NONE, AGT_INPUT_CAPTURE_SIGNAL_FILTER_1, AGT_INPUT_CAPTURE_SIGNAL_FILTER_8, AGT_INPUT_CAPTURE_SIGNAL_FILTER_32 }

enum	agt_input_capture_clock_divider_t {
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_1,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_2,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_4,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_8,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_16,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_32,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_64,
	AGT_INPUT_CAPTURE_CLOCK_DIVIDER_128
	}

enum	agt_input_capture_event_flag_t { AGT_INPUT_CAPTURE_ACTIVE_EDGE_FLAG = 16U, AGT_INPUT_CAPTURE_UNDERFLOW_FLAG = 32U, AGT_INPUT_CAPTURE_COMPARE_A_FLAG = 64U, AGT_INPUT_CAPTURE_COMPARE_B_FLAG = 128U }
enum	agt_input_capture_mode_t { AGT_INPUT_CAPTURE_MODE_PULSE_WIDTH = 3U, AGT_INPUT_CAPTURE_MODE_PERIOD = 4U, AGT_INPUT_CAPTURE_MODE_PULSE_COUNT = 2U }
enum	agt_input_capture_pin_select_t { AGT_INPUT_CAPTURE_PIN_AGTIO_A = 0U, AGT_INPUT_CAPTURE_PIN_AGTIO_B = 2U, AGT_INPUT_CAPTURE_PIN_AGTIO_C = 3U }

Functions

ssp_err_t	R_AGT_InputCaptureOpen (input_capture_ctrl_t *const p_api_ctrl, input_capture_cfg_t const *const p_cfg) Open an AGT Timer for Input Capture. Implements input_capture_api_t::open. More...
ssp_err_t	R_AGT_InputCaptureClose (input_capture_ctrl_t *const p_api_ctrl) Close a AGT Timer Channel for Input Capture. Implements input_capture_api_t::close. More...

ssp_err_t	R_AGT_InputCaptureVersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. Implements input_capture_api_t::versionGet. More...
ssp_err_t	R_AGT_InputCaptureDisable (input_capture_ctrl_t const *const p_api_ctrl) Stops the Input capture and disables its interrupts for specified channel at NVIC. Implements input_capture_api_t::disable. More...
ssp_err_t	R_AGT_InputCaptureEnable (input_capture_ctrl_t const *const p_api_ctrl) Enables its interrupts for specified channel at NVIC, and starts the Input capture. Implements input_capture_api_t::enable. More...
ssp_err_t	R_AGT_InputCaptureInfoGet (input_capture_ctrl_t const *const p_api_ctrl, input_capture_info_t *const p_info) Gets status into provided p_info pointer. Implements input_capture_api_t::infoGet. More...
ssp_err_t	R_AGT_InputCaptureLastCaptureGet (input_capture_ctrl_t const *const p_api_ctrl, input_capture_capture_t *const p_capture) Update the last captured value and overflow count, in provided p_capture pointer. Implements input_capture_api_t::lastCaptureGet. More...

Detailed Description

Driver for the Asynchronous General-Purpose Timer (AGT) with Input Capture.

Summary

Extends Input Capture Interface.

This module implements the Input Capture Interface for the Asynchronous General-Purpose Timer (AGT) peripherals.

Enumeration Type Documentation

◆ agt_input_capture_clock_divider_t

enum agt_input_capture_clock_divider_t	
AGT Input capture AGT LOCO or AGT FSUB divider.	
Enumerator	
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_1	/ 1
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_2	/ 2
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_4	/ 4
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_8	/ 8
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_16	/ 16
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_32	/ 32
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_64	/ 64
AGT_INPUT_CAPTURE_CLOCK_DIVIDER_128	/ 128

◆ agt_input_capture_count_edges_t

enum agt_input_capture_count_edges_t	
AGT Input capture signal edge polarity for event counter mode	
Enumerator	
INPUT_CAPTURE_SIGNAL_SINGLE_EDGE	Counts only one edge of the pulse.
INPUT_CAPTURE_SIGNAL_BOTH_EDGE	Counts both edges of the pulse.

◆ agt_input_capture_count_source_t

enum agt_input_capture_count_source_t	
Count source	
Enumerator	
AGT_INPUT_CAPTURE_CLOCK_PCLKB	Clock AGT_CLOCK_PCLKB.
AGT_INPUT_CAPTURE_CLOCK_PCLKB_DIV_8	Superseded: See AGT_CLOCK_PCLKB.
AGT_INPUT_CAPTURE_CLOCK_PCLKB_DIV_2	Superseded: See AGT_CLOCK_PCLKB.
AGT_INPUT_CAPTURE_CLOCK_LOCO	Divided clock LOCO specified by bits CKS[2:0] in the AGTMR2 register.
AGT_CLOCK_INPUT_CAPTURE_FSUB	Divided clock fSUB specified by bits CKS[2:0] in the AGTMR2 register.

◆ agt_input_capture_event_flag_t

enum agt_input_capture_event_flag_t	
AGT Input capture Event flags.	
Enumerator	
AGT_INPUT_CAPTURE_ACTIVE_EDGE_FLAG	Measurement event flag.
AGT_INPUT_CAPTURE_UNDERFLOW_FLAG	Underflow event flag.
AGT_INPUT_CAPTURE_COMPARE_A_FLAG	Compare match A event flag.
AGT_INPUT_CAPTURE_COMPARE_B_FLAG	Compare match B event flag.

◆ agt_input_capture_mode_t

enum agt_input_capture_mode_t	
AGT Input capture modes.	
Enumerator	
AGT_INPUT_CAPTURE_MODE_PULSE_WIDTH	Measure a signal pulse width.
AGT_INPUT_CAPTURE_MODE_PERIOD	Measure a signal cycle period.
AGT_INPUT_CAPTURE_MODE_PULSE_COUNT	Measure a signal event count.

◆ agt_input_capture_pin_select_t

enum agt_input_capture_pin_select_t	
AGT Input capture AGTIO Pin Select.	
Enumerator	
AGT_INPUT_CAPTURE_PIN_AGTIO_A	Selects the pin AGTIO_A for input capture.
AGT_INPUT_CAPTURE_PIN_AGTIO_B	Selects the pin AGTIO_B for input capture.
AGT_INPUT_CAPTURE_PIN_AGTIO_C	Selects the pin AGTIO_C for input capture.

◆ agt_input_capture_signal_filter_t

enum agt_input_capture_signal_filter_t	
Input capture signal noise filter (debounce) setting. Only available for input signals in AGTIO pins. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, AGT section.	
Enumerator	
AGT_INPUT_CAPTURE_SIGNAL_FILTER_NONE	NO FILTER.
AGT_INPUT_CAPTURE_SIGNAL_FILTER_1	PCLK/1.
AGT_INPUT_CAPTURE_SIGNAL_FILTER_8	PCLK/8.
AGT_INPUT_CAPTURE_SIGNAL_FILTER_32	PCLK/32.

Function Documentation

◆ R_AGT_InputCaptureClose()

ssp_err_t R_AGT_InputCaptureClose (input_capture_ctrl_t *const *p_api_ctrl*)

Close a AGT Timer Channel for Input Capture. Implements input_capture_api_t::close.

Clears Timer settings, disables interrupts, and clears internal driver data.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Cleanup. Disable interrupts and stop measurements.

Unlock channel

Clear stored internal driver data

◆ R_AGT_InputCaptureDisable()

ssp_err_t R_AGT_InputCaptureDisable (input_capture_ctrl_t const *const *p_api_ctrl*)

Stops the Input capture and disables its interrupts for specified channel at NVIC. Implements input_capture_api_t::disable.

Return values

SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Disable interrupts

◆ R_AGT_InputCaptureEnable()

ssp_err_t R_AGT_InputCaptureEnable (input_capture_ctrl_t const *const *p_api_ctrl*)

Enables its interrupts for specified channel at NVIC, and starts the Input capture. Implements input_capture_api_t::enable.

Return values

SSP_SUCCESS	Interrupt enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Enabling the measurement overflow and compare match interrupt.

◆ R_AGT_InputCaptureInfoGet()

ssp_err_t R_AGT_InputCaptureInfoGet (input_capture_ctrl_t const *const *p_api_ctrl*, input_capture_info_t *const *p_info*)

Gets status into provided p_info pointer. Implements input_capture_api_t::infoGet.

Return values

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Gets the input capture status.

◆ R_AGT_InputCaptureLastCaptureGet()

ssp_err_t R_AGT_InputCaptureLastCaptureGet (input_capture_ctrl_t const *const *p_api_ctrl*, input_capture_capture_t *const *p_capture*)

Update the last captured value and overflow count, in provided p_capture pointer. Implements input_capture_api_t::lastCaptureGet.

Return values

SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Gets the captured value

◆ R_AGT_InputCaptureOpen()

ssp_err_t R_AGT_InputCaptureOpen (input_capture_ctrl_t *const *p_api_ctrl*, input_capture_cfg_t const *const *p_cfg*)

Open an AGT Timer for Input Capture. Implements input_capture_api_t::open.

The Open function configures a single AGT channel for input capture and provides a handle for use with the other Input Capture API functions. This function must be called once prior to calling any other Input Capture API function. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the parameters is NULL: p_cfg, p_ctrl, p_extend.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Get fmi feature information for AGT.

If count source clock is not operational, return error.

Verify channel is not already used

Get fmi measurement and overflow event information for AGT.

Set measurement and overflow event interrupt priority and vector info.

Get fmi compare match event information for AGT.

Set compare match event interrupt priority and vector info.

Initialize control block.

Perform hardware initializations based on configuration.

Mark channel as open, by initializing it to R_AIC ASCII equivalent.

◆ R_AGT_InputCaptureVersionGet()

ssp_err_t R_AGT_InputCaptureVersionGet (ssp_version_t *const *p_version*)
Gets driver version based on compile time macros. Implements input_capture_api_t::versionGet. **Return values** SSP_SUCCESS — Success. SSP_ERR_ASSERTION — The parameter p_version is NULL.

agt_input_capture_extend_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » AGT Input Capture

Extension configuration struct for AGT Input Capture. More...

#include <r_agt_input_capture.h>

Data Fields

uint16_t	pulse_count_value Selects the pulse count value for pulse count capture.
agt_input_capture_signal_filter_t	signal_filter Selects the input signal filter.
agt_input_capture_count_source_t	count_source Selects the input count clock source.
agt_input_capture_clock_divider_t	clock_divider Selects the AGT LOCO or AGT FSUB divider.
agt_input_capture_count_edges_t	count_edge Selects the count edge for pulse count capture.

agt_input_capture_pin_select_t	pin_select Selects the pin for agt input capture.

Detailed Description

Extension configuration struct for AGT Input Capture.

Pointed to by input_capture_cfg_t.p_extend

The documentation for this struct was generated from the following file:

- r_agt_input_capture.h

agt_input_capture_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » AGT Input Capture

#include <r_agt_input_capture.h>

Data Fields

uint32_t	open Whether or not channel is open.
uint8_t	channel The channel in use.
uint8_t	flags Input capture Event flags.
input_capture_mode_t	mode The mode of measurement being performed.
input_capture_repetition_t	repetition One-shot or periodic measurement.

volatile uint32_t	capture_count The value of the timer captured at the time of interrupt.
volatile uint32_t	overflows_current Running count of overflows in current measurement.
void(*	p_callback)(input_capture_callback_args_t *p_args) Pointer to user callback.
void const *	p_context Pointer to user's context data, to be passed to the callback function.
void *	p_reg AGT base register for this channel.
IRQn_Type	capture_irq Capture IRQ number.
IRQn_Type	overflow_irq Overflow IRQ number.
volatile bool	pulse_period_first_edge Whether the first edge in period has received.

Detailed Description

Channel control block. DO NOT INITIALIZE. Initialization occurs when input_capture_api_t::open is called.

The documentation for this struct was generated from the following file:

- r_agt_input_capture.h

5.1.5.6 Analog Connections

Renesas Synergy Software Package Reference » HAL Layer

Driver for internal analog connections. More...

Functions

ssp_err_t	R_ANALOG_CONNECT_Init (analog_connect_cfg_t const *const p_cfg)
ssp_err_t	R_ANALOG_CONNECT_Connect (analog_connect_t const connection)
ssp_err_t	R_ANALOG_CONNECT_ConnectMultiple (analog_connect_table_t const *const p_table)
ssp_err_t	R_ANALOG_CONNECT_VersionGet (ssp_version_t *const p_version)

Detailed Description

Driver for internal analog connections.

Summary

Extends Analog Connect Interface.

This module implements the Analog Connect Interface.

Function Documentation

◆ R_ANALOG_CONNECT_Connect()

ssp_err_t R_ANALOG_CONNECT_Connect (analog_connect_t const *connection*)

Makes an internal analog connection.

For ACMPHS connections, output and interrupts are disabled while the connection is configured, then the output and interrupts are restored to their original state. Since this function enables output if it was enabled before, R_ACMPHS_Close() should not be called during this function.

ACMPLP connections must be made prior to enabling output.

OPAMP connections can be reconfigured while the OPAMP is operating. If an OPAMP connection is already set and a new connection is made, the new connection overwrites the existing connection. OPAMP connections can be OR'd together if they are for the same signal:

- Valid example: ANALOG_CONNECT_OPAMP0_PLUS_TO_PIN_AMP0_PLUS | ANALOG_CONNECT_OPAMP0_PLUS_TO_PIN_AMP1_PLUS
  - Both start with ANALOG_CONNECT_OPAMP0_PLUS
- Invalid example: ANALOG_CONNECT_OPAMP0_PLUS_TO_PIN_AMP0_PLUS | ANALOG_CONNECT_OPAMP0_MINUS_TO_PIN_AMP0_MINUS
  - Do not mix PLUS and MINUS
- Invalid example: ANALOG_CONNECT_OPAMP0_PLUS_TO_PIN_AMP0_PLUS | ANALOG_CONNECT_OPAMP1_PLUS_TO_PIN_AMP1_PLUS
  - Do not mix channels 0 and 1

If AVCC0 < 2.7 V, MOCO must be enabled to make OPAMP connections because the internal OPAMP switches require a charge pump, and the MOCO is required for charge pump operation. When using the charge pump for the amplifier:

- Turn on no more than a total of 5 connections for OPAMP0.
- Turn on no more than a total of 5 connections for OPAMP1.
- Turn on no more than a total of 2 connections for OPAMP2.

Return values

SSP_SUCCESS	Connection made.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- fmi_api_t::productFeatureGet

◆ R_ANALOG_CONNECT_ConnectMultiple()

ssp_err_t R_ANALOG_CONNECT_ConnectMultiple (analog_connect_table_t const *const *p_table*)

Makes all connections in the table.

See R_ANALOG_CONNECT_Connect() for modules specific usage notes regarding connections.

Return values

SSP_SUCCESS	All connections made.
SSP_ERR_ASSERTION	Data table pointer is NULL or size is 0.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

◆ R_ANALOG_CONNECT_Init()

ssp_err_t R_ANALOG_CONNECT_Init (analog_connect_cfg_t const *const *p_cfg*)

Placeholder function for analog connection initialization code. Currently unused.

Return values

SSP_SUCCESS	Init successful.

◆ R_ANALOG_CONNECT_VersionGet()

ssp_err_t R_ANALOG_CONNECT_VersionGet (ssp_version_t *const *p_version*)

Gets the API and code version. Implements analog_connect_api_t::versionGet().

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

5.1.5.7 CAC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Clock Frequency Accuracy Measurement Circuit (CAC). More...

Data Structures

struct	cac_instance_ctrl_t

Functions

ssp_err_t	R_CAC_Open (cac_ctrl_t *const p_api_ctrl, cac_cfg_t const *const p_cfg) Initialize the CAC peripheral. More...
ssp_err_t	R_CAC_Close (cac_ctrl_t *const p_api_ctrl) Release any resources that were allocated by the Open() or any subsequent CAC operations. Implements r_cac_t::close. More...
ssp_err_t	R_CAC_StopMeasurement (cac_ctrl_t *const p_api_ctrl) Stop the CAC measurement process. Implements r_cac_t::stopMeasurement. More...
ssp_err_t	R_CAC_StartMeasurement (cac_ctrl_t *const p_api_ctrl) Start the CAC measurement process. Implements r_cac_t::startMeasurement. More...
ssp_err_t	R_CAC_Reset (cac_ctrl_t *const p_api_ctrl) Resets the Overflow, Measurement End and Frequency Error interrupt flags. This will clear any of the CASTR status bits that have been set, but only if the CFME bit is off (Not measuring). Implements r_cac_t::reset. More...
ssp_err_t	R_CAC_Read (cac_ctrl_t *const p_api_ctrl, uint8_t *const p_status, uint16_t *const p_counter) Read and return the CAC status and counter registers. Implements r_cac_t::read. More...
ssp_err_t	R_CAC_VersionGet (ssp_version_t *const p_version) Get the API and code version information. More...

Detailed Description

Driver for the Clock Frequency Accuracy Measurement Circuit (CAC).

Summary

This module supports the CAC peripheral.

Function Documentation

◆ R_CAC_Close()

ssp_err_t R_CAC_Close (cac_ctrl_t *const *p_ctrl*)

Release any resources that were allocated by the Open() or any subsequent CAC operations. Implements r_cac_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Define a control block structure used in the API calls.
cac_ctrl_t ctrl;
err = R_CAC_Close(&ctrl);
```

Eliminate warning if parameter checking is disabled.

Disable interrupts in the peripheral and NVIC

Disable interrupts in NVIC.

Disable the CAC ints.

Stop measuring.

Power down peripheral.

Return the hardware lock for the CAC.

◆ R_CAC_Open()

ssp_err_t R_CAC_Open (cac_ctrl_t *const *p_ctrl*, cac_cfg_t const *const *p_cfg*)

Initialize the CAC peripheral.

The Open function applies power to the CAC peripheral, checks/sets the interrupt priority, and configures the CAC based on the provided user configuration settings. If a user defined callback function has been provided in the configuration, then the CAC interrupt(s) will be enabled and the user callback function called accordingly. Implements r_cac_t::open.

Return values

SSP_SUCCESS	CAC is available and available for measurement(s).
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options are invalid.
SSP_ERR_HW_LOCKED	Hardware lock for CAC peripheral is already taken.
SSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate smaller than reference clock rate.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

There is only a single CAC peripheral. It is not reentrant.

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Define a control block structure used in the API calls.
cac_ctrl_t ctrl;
// Define a configuration structure used in the API calls.
cac_cfg_t g_cac_example_cfg;
// Init the CGC and start the HOCO and LOCO clocks.
err = R_CGC_Init();
err = R_CGC_ClockStart(CGC_CLOCK_HOCO, &clock_config);
err = R_CGC_ClockStart(CGC_CLOCK_LOCO, &clock_config);

// Specify the parameters we are using
g_cac_example_cfg.callback = NULL,
```

```
g_cac_example_cfg.p_context = 0,
g_cac_example_cfg.p_extend = NULL,
g_cac_example_cfg.continuous_mode = false,       // measurement does not continuously
restart after completing.
g_cac_example_cfg.mei_interrupt_enabled = false; // Measurement complete interrupt
disabled
g_cac_example_cfg.ovf_interrupt_enabled = false; // Overflow interrupt is disabled
g_cac_example_cfg.ferr_interrupt_enabled = false; // Frequency Error interrupt is
disabled
g_cac_example_cfg.cac_ref_clock.digfilter = CAC_REF_DIGF_OFF; // No digital filter
g_cac_example_cfg.cac_ref_clock.edge = CAC_REF_EDGE_RISE;      // Rising edge detect
g_cac_example_cfg.cac_meas_clock.clock = CAC_CLOCK_SOURCE_HOCO; // We want to measure
HOCO (24 MHz)
g_cac_example_cfg.cac_meas_clock.divider = CAC_MEAS_DIV_1;       // No divisor on the
measurement clock
g_cac_example_cfg.cac_ref_clock.clock = CAC_CLOCK_SOURCE_LOCO;   // Our reference
clock will LOCO (32.768 kHz)
g_cac_example_cfg.cac_ref_clock.divider = CAC_REF_DIV_32;       // Minimum divider is
32, so effective freq = 1024 Hz
 err = R_CAC_Open(&ctrl, &g_cac_example_cfg);
```

g_cac_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that they are not accessed. The code below eliminates this warning and also ensures these data structures are not optimised away.

Eliminate warning if parameter checking is disabled.

Take the hardware lock for the CAC.

Setup the interrupt vectors and priorities

Return the hardware lock for the CAC.

Apply power to the peripheral

Configure the CAC per the configuration.

Store the callback and context information

Mark driver as open by initializing it to "CAC" - its ASCII equivalent.

◆ R_CAC_Read()

ssp_err_t R_CAC_Read (cac_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*, uint16_t *const *p_counter*)

Read and return the CAC status and counter registers. Implements r_cac_t::read.

Return values

SSP_SUCCESS	CAC read successful.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

```
// Example code
// Create a variable for the error and storage for the CAC count and status
ssp_err_t err;
uint8_t   cac_status;
uint16_t cac_counter;
// Define a control block structure used in the API calls.
cac_ctrl_t ctrl;
err = R_CAC_Read(&ctrl, &cac_status, &cac_counter);
```

Eliminate warning if parameter checking is disabled.

◆ R_CAC_Reset()

ssp_err_t R_CAC_Reset (cac_ctrl_t *const *p_ctrl*)

Resets the Overflow, Measurement End and Frequency Error interrupt flags. This will clear any of the CASTR status bits that have been set, but only if the CFME bit is off (Not measuring). Implements r_cac_t::reset.

Return values

SSP_SUCCESS	CAC reset completed.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Define a control block structure used in the API calls.
 cac_ctrl_t ctrl;
// Start the measurement process
err = R_CAC_Reset(&ctrl);
```

Eliminate warning if parameter checking is disabled.

Reset the CAC.

◆ R_CAC_StartMeasurement()

ssp_err_t R_CAC_StartMeasurement (cac_ctrl_t *const *p_ctrl*)

Start the CAC measurement process. Implements r_cac_t::startMeasurement.

Return values

SSP_SUCCESS	CAC measurement started.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.
SSP_ERR_CLOCK_INACTIVE	Either the provided Measurement or Reference clock is not running

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- cgc_api_t::clockCheck

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Define a control block structure used in the API calls.
 cac_ctrl_t ctrl;
// Start the measurement process
err = R_CAC_StartMeasurement(&ctrl);
```

Eliminate warnings if parameter checking is disabled.

Start measuring.

◆ R_CAC_StopMeasurement()

ssp_err_t R_CAC_StopMeasurement (cac_ctrl_t *const *p_ctrl*)

Stop the CAC measurement process. Implements r_cac_t::stopMeasurement.

Return values

SSP_SUCCESS	CAC measuring has been stopped.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Define a control block structure used in the API calls.
 cac_ctrl_t ctrl;
// Stop the measurement process
err = R_CAC_StopMeasurement(&ctrl);
```

Eliminate warning if parameter checking is disabled.

Stop measuring.

◆ R_CAC_VersionGet()

ssp_err_t R_CAC_VersionGet (ssp_version_t *const *p_version*)
Get the API and code version information. **Return values** SSP_SUCCESS — Version info returned. ***Note*** *This function is reentrant.*

```
// Example code
// Create a variable for the error
ssp_err_t err;
// Create a variable for the version information
ssp_version_t version;
// Get the current counter value
err = R_CAC_VersionGet(&version);
```

cac_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CAC

#include <r_cac.h>

Data Fields

void *	p_reg Pointer to register base address.
void(*	p_callback)(cac_callback_args_t *cb_data) Called from the ISR.
void const *	p_context Passed to the callback.
IRQn_Type	frequency_error_irq Frequency error IRQ number.

IRQn_Type	measurement_end_irq Measurement end IRQ number.
IRQn_Type	overflow_irq Overflow IRQ number.
uint32_t	cac_api_open Set to "CAC" once API has been successfully opened.
bool	cac_continous_mode Set as a result of the Open() call.
bsp_lock_t	cac_lock CAC commands software lock.
cac_clock_source_t	measurement_clock Clock specified in Open() as the measurement clock.
cac_clock_source_t	reference_clock Clock specified in Open() as the reference clock.

Detailed Description

CAC instance control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_cac.h

5.1.5.8 CAN

Renesas Synergy Software Package Reference » HAL Layer

Driver for CAN, Controller Area Network. More...

Data Structures

struct	can_instance_ctrl_t
struct	can_extended_cfg_t

Functions

ssp_err_t	R_CAN_Open (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg) Open and configure the CAN channel for operation. Implements can_api_t::open() More...
ssp_err_t	R_CAN_Close (can_ctrl_t *const p_ctrl) Close the CAN channel. Implements can_api_t::close() More...
ssp_err_t	R_CAN_Read (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame) Read data from the CAN channel. Return up to eight bytes read from the channel mailbox. Implements can_api_t::read() More...
ssp_err_t	R_CAN_Write (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame) Write data to the CAN channel. Write up to eight bytes to the channel mailbox. Implements can_api_t::write() More...
ssp_err_t	R_CAN_Control (can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data) CAN Control is used to control extended features. Implements can_api_t::control() More...
ssp_err_t	R_CAN_InfoGet (can_ctrl_t *const p_ctrl, can_info_t *const p_info) Get CAN state and status information for the channel. Implements can_api_t::infoGet() More...
ssp_err_t	R_CAN_VersionGet (ssp_version_t *const p_version) Get CAN module code and API versions. Implements can_api_t::versionGet() More...

Detailed Description

Driver for CAN, Controller Area Network.

This module supports the Controller Area Network peripheral. It implements the following interfaces:

- CAN Interface

Function Documentation

◆ R_CAN_Close()

ssp_err_t R_CAN_Close (can_ctrl_t *const *p_ctrl*)

Close the CAN channel. Implements can_api_t::close()

Return values

SSP_SUCCESS	Channel closed successfully.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_ASSERTION	Null pointer presented.

Mark the channel not open so other APIs cannot use it.

Disable transmit, receive and error interrupts

Enable module stop for the CAN channel

Unlock the CAN channel

◆ R_CAN_Control()

ssp_err_t R_CAN_Control (can_ctrl_t *const *p_ctrl*, can_command_t const *command*, void * *p_data*)

CAN Control is used to control extended features. Implements can_api_t::control()

Return values

SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_INVALID_ARGUMENT	Invalid command.
SSP_ERR_ASSERTION	Null pointer presented
SSP_ERR_CAN_MODE_SWITCH_FAILED	Switching modes failed.

Verify command is CAN_COMMAND_MODE_SWITCH

Change operating mode. Returns false if invalid mode or mode switch failed.

Save mode for diagnostic purposes.

◆ R_CAN_InfoGet()

ssp_err_t R_CAN_InfoGet (can_ctrl_t *const *p_ctrl*, can_info_t *const *p_info*)

Get CAN state and status information for the channel. Implements can_api_t::infoGet()

Return values

SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	Channel failed to return info.
SSP_ERR_ASSERTION	Null pointer presented

Get status for channel.

Error encountered when retrieving info.

Save the operation mode

◆ R_CAN_Open()

ssp_err_t R_CAN_Open (can_ctrl_t *const *p_ctrl*, can_cfg_t const *const *p_cfg*)

Open and configure the CAN channel for operation. Implements can_api_t::open()

Return values

SSP_SUCCESS	Channel opened successfully
SSP_ERR_INVALID_ARGUMENT	Invalid channel passed as argument.
SSP_ERR_HW_LOCKED	Lock already owned by another user.
SSP_ERR_CAN_MODE_SWITCH_FAILED	Channel failed to switch modes.
SSP_ERR_CAN_INIT_FAILED	Channel failed to initialize.
SSP_ERR_ASSERTION	Null pointer presented.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- cgc_api_t::clockCheck

Check for valid parameters.

Make sure the feature exists on this MCU.

Return if failed to get feature information.

Try to get channel lock.

Return if channel is already open so return error

Enter module start state.

Disable interrupts while initializing

Initialize and configure CAN module to run.

Set channel, callback function, context, id mode, mailbox count, message mode, op mode and opened status.

If successful, Lookup and store IRQ numbers. Enable interrupts.

If successful, Mark the control block as open

If the device failed to initialize, disable interrupts, stop and unlock the hardware and mark the control block as closed.

Process errors before returning.

Log error or assertion.

◆ R_CAN_Read()

ssp_err_t R_CAN_Read (can_ctrl_t *const *p_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame*)

Read data from the CAN channel. Return up to eight bytes read from the channel mailbox. Implements can_api_t::read()

Return values

SSP_SUCCESS	Data successfully read.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	No data available.
SSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is not setup for receive.
SSP_ERR_ASSERTION	Null pointer presented.

Check for receive data

Get frame data.

Check for other mailboxes in an overrun state.

Check for other mailboxes with received messages pending.

◆ R_CAN_VersionGet()

ssp_err_t R_CAN_VersionGet (ssp_version_t *const *p_version*)

Get CAN module code and API versions. Implements can_api_t::versionGet()

Return values

SSP_SUCCESS	Operation succeeded.
SSP_ERR_ASSERTION	Null pointer presented note This function is reentrant.

Return module version information.

◆ R_CAN_Write()

ssp_err_t R_CAN_Write (can_ctrl_t *const *p_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame*)

Write data to the CAN channel. Write up to eight bytes to the channel mailbox. Implements can_api_t::write()

Return values

SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
SSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup for receive and cannot send.
SSP_ERR_INVALID_ARGUMENT	Data length or frame type invalid.
SSP_ERR_ASSERTION	Null pointer presented

Check transmit ready flag.

Transmit ready flag is not set, return error/status.

Transmit ready flag set, so clear it.

Send transmit frame.

can_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CAN

#include <r_can.h>

Data Fields

uint32_t	channel Channel number. More...
uint32_t	open Open status of channel.
can_mode_t	operation_mode Can operation mode.

can_id_mode_t	id_mode Standard or Extended ID mode.
uint32_t	mailbox_count Number of mailboxes.
can_mailbox_t *	p_mailbox Pointer to mailboxes.
can_message_mode_t	message_mode Overwrite message or overrun.
can_clock_source_t	clock_source Clock source. CANMCLK or PCLKB.
void(*	p_callback)(can_callback_args_t *p_args) Pointer to callback function. More...
void const *	p_context Pointer to the higher level device context.
void *	p_reg Pointer to register base address.
IRQn_Type	error_irq Error IRQ number.
IRQn_Type	mailbox_rx_irq Receive mailbox IRQ number.
IRQn_Type	mailbox_tx_irq Transmit mailbox IRQ number.

Detailed Description

CAN Instance Control Block

Field Documentation

◆ channel

uint32_t can_instance_ctrl_t::channel
Channel number. Parameters to control CAN peripheral device

◆ p_callback

void(* can_instance_ctrl_t::p_callback) (can_callback_args_t *p_args)
Pointer to callback function. Parameters to process CAN Event

The documentation for this struct was generated from the following file:

- r_can.h

can_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CAN

#include <r_can.h>

Data Fields

can_clock_source_t	clock_source Source of the CAN clock.
uint32_t *	p_mailbox_mask Mailbox mask, one for every 4 mailboxes.

Detailed Description

CAN clock configuration and mailbox mask to be pointed to by p_extend.

The documentation for this struct was generated from the following file:

- r_can.h

5.1.5.9 CGC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Clock Generation Circuit. More...

Functions

ssp_err_t	R_CGC_Init (void) Initialize the CGC API. More...
ssp_err_t	R_CGC_ClocksCfg (cgc_clocks_cfg_t const *const p_clock_cfg) Reconfigure all main system clocks. More...
ssp_err_t	R_CGC_ClockStart (cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg) Start the specified clock if it is not currently active. More...
ssp_err_t	R_CGC_ClockStop (cgc_clock_t clock_source) Stop the specified clock if it is active and not configured as the system clock. More...
ssp_err_t	R_CGC_SystemClockSet (cgc_clock_t clock_source, cgc_system_clock_cfg_t const *const p_clock_cfg) Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD and FCLK. More...
ssp_err_t	R_CGC_SystemClockGet (cgc_clock_t *clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg) Return the current system clock source and configuration. More...
ssp_err_t	R_CGC_SystemClockFreqGet (cgc_system_clocks_t clock, uint32_t

	*p_freq_hz) Return the requested internal clock frequency in Hz. More...
ssp_err_t	R_CGC_ClockCheck (cgc_clock_t clock_source) Check the specified clock for stability. More...
ssp_err_t	R_CGC_OscStopDetect (void(*p_callback)(cgc_callback_args_t *p_args), bool enable) Enable or disable the oscillation stop detection for the main clock. The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency. More...
ssp_err_t	R_CGC_OscStopStatusClear (void) Clear the Oscillation Stop Detection Status register. More...
ssp_err_t	R_CGC_BusClockOutCfg (cgc_bclockout_dividers_t divider) Configure the secondary dividers for BCLKOUT. The primary divider is set using the bsp clock configuration and the R_CGC_SystemClockSet function. More...
ssp_err_t	R_CGC_BusClockOutEnable (void) Enable the BCLKOUT output. More...
ssp_err_t	R_CGC_BusClockOutDisable (void) Disable the BCLKOUT output. More...
ssp_err_t	R_CGC_ClockOutCfg (cgc_clock_t clock, cgc_clockout_dividers_t divider) Configure the dividers for CLKOUT. More...
ssp_err_t	R_CGC_ClockOutEnable (void) Enable the CLKOUT output. More...
ssp_err_t	R_CGC_ClockOutDisable (void) Disable the CLKOUT output. More...

ssp_err_t	R_CGC_LCDClockCfg (cgc_clock_t clock) Configure the source for the segment LCDCLK. More...
ssp_err_t	R_CGC_LCDClockEnable (void) Enable the segment LCDCLK output. More...
ssp_err_t	R_CGC_LCDClockDisable (void) Disable the segment LCDCLK output. More...
ssp_err_t	R_CGC_SDADCClockCfg (cgc_clock_t clock) Configure the source for the SDADCCLK. More...
ssp_err_t	R_CGC_SDADCClockEnable (void) Enable the SDADCCLK output. More...
ssp_err_t	R_CGC_SDADCClockDisable (void) Disable the SDADCCLK output. More...
ssp_err_t	R_CGC_SDRAMClockOutEnable (void) Enable the SDCLK output. More...
ssp_err_t	R_CGC_SDRAMClockOutDisable (void) Disable the SDCLK output. More...
ssp_err_t	R_CGC_USBClockCfg (cgc_usb_clock_div_t divider) Configure the dividers for UCLK. More...
ssp_err_t	R_CGC_SystickUpdate (uint32_t period_count, cgc_systick_period_units_t units) Re-Configure the systick based on the provided period and current system clock frequency. More...
ssp_err_t	R_CGC_VersionGet (ssp_version_t *const p_version) Return the driver version. More...

bool	r_cgc_clock_run_state_get (R_SYSTEM_Type *p_system_reg, cgc_clock_t clock) This function returns the run state of the selected clock. More...
cgc_operating_modes_t	r_cgc_operating_mode_get (R_SYSTEM_Type *p_system_reg) This function checks the MCU for High Speed Mode. More...
void	r_cgc_operating_hw_modeset (R_SYSTEM_Type *p_system_reg, cgc_operating_modes_t operating_mode) This function changes the operating power control mode. More...
void	r_cgc_hoco_wait_control_set (R_SYSTEM_Type *p_system_reg, uint8_t hoco_wait) This function sets the HOCO wait time register. More...

Detailed Description

Driver for the Clock Generation Circuit.

Clock Generation Circuit Hardware Functions.

This module supports the Clock Generation Circuit. It implements the following interfaces:

- CGC Interface

Function Documentation

◆ R_CGC_BusClockOutCfg()

ssp_err_t R_CGC_BusClockOutCfg (cgc_bclockout_dividers_t *divider*)

Configure the secondary dividers for BCLKOUT. The primary divider is set using the bsp clock configuration and the R_CGC_SystemClockSet function.

Return values

SSP_SUCCESS	Operation performed successfully.

◆ R_CGC_BusClockOutDisable()

ssp_err_t R_CGC_BusClockOutDisable (void)

Disable the BCLKOUT output.

Return values

SSP_SUCCESS	Operation performed successfully.

◆ R_CGC_BusClockOutEnable()

ssp_err_t R_CGC_BusClockOutEnable (void)

Enable the BCLKOUT output.

Return values

SSP_SUCCESS	Operation performed successfully.

◆ r_cgc_clock_run_state_get()

bool r_cgc_clock_run_state_get (R_SYSTEM_Type * *p_system_reg*, cgc_clock_t *clock*)

This function returns the run state of the selected clock.

Parameters

[in]	clock	the clock to check
[in]	p_system_reg	pointer to system register structure
[in]	clock	- the clock to check

Return values

bool	true if clock is running, false if stopped

◆ R_CGC_ClockCheck()

ssp_err_t R_CGC_ClockCheck (cgc_clock_t *clock_source*)

Check the specified clock for stability.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_NOT_STABILIZED	Clock not stabilized.
SSP_ERR_CLOCK_ACTIVE	Clock active but not able to check for stability.
SSP_ERR_CLOCK_INACTIVE	Clock not turned on.
SSP_ERR_INVALID_ARGUMENT	Illegal parameter passed.
SSP_ERR_STABILIZED	Clock stabilized.

◆ R_CGC_ClockOutCfg()

ssp_err_t R_CGC_ClockOutCfg (cgc_clock_t *clock*, cgc_clockout_dividers_t *divider*)

Configure the dividers for CLKOUT.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	return error if PLL is used as source for clock out
SSP_ERR_CLOCK_INACTIVE	return error if sub clock is not started prior to using it for clock out

◆ R_CGC_ClockOutDisable()

ssp_err_t R_CGC_ClockOutDisable (void)

Disable the CLKOUT output.

Return values

SSP_SUCCESS	Operation performed successfully.

◆ R_CGC_ClockOutEnable()

ssp_err_t R_CGC_ClockOutEnable (void)

Enable the CLKOUT output.

Return values

SSP_SUCCESS	Operation performed successfully.

◆ R_CGC_ClocksCfg()

ssp_err_t R_CGC_ClocksCfg (cgc_clocks_cfg_t const *const *p_clock_cfg*)

Reconfigure all main system clocks.

Return values

SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_MAIN_OSC_INACTIVE	PLL Initialization attempted with Main OCO turned off/unstable.
SSP_ERR_CLOCK_ACTIVE	Active clock source specified for modification. This applies specifically to the PLL dividers/multipliers which cannot be modified if the PLL is active. It has to be stopped first before modification.
SSP_ERR_NOT_STABILIZED	The Clock source is not stabilized after being turned off.
SSP_ERR_CLKOUT_EXCEEDED	The main oscillator can be only 8 or 16 MHz.
SSP_ERR_ASSERTION	A NULL is passed for configuration data when PLL is the clock_source.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.

◆ R_CGC_ClockStart()

ssp_err_t R_CGC_ClockStart (cgc_clock_t *clock_source*, cgc_clock_cfg_t * *p_clock_cfg*)

Start the specified clock if it is not currently active.

Configures the following when starting the Main Clock Oscillator:

- MainClock drive capacity (Configured based on external clock frequency)
- MainClock stabilization wait time (Compile time configurable: CGC_CFG_MAIN_OSC_WAIT)
- To update the subclock driven capacity, stop the subclock first before calling this function.

Return values

SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_MAIN_OSC_INACTIVE	PLL Initialization attempted with Main OCO turned off/unstable.
SSP_ERR_CLOCK_ACTIVE	Active clock source specified for modification. This applies specifically to the PLL dividers/multipliers which cannot be modified if the PLL is active. It has to be stopped first before modification.
SSP_ERR_NOT_STABILIZED	The Clock source is not stabilized after being turned off.
SSP_ERR_CLKOUT_EXCEEDED	The main oscillator can be only 8 or 16 MHz.
SSP_ERR_ASSERTION	A NULL is passed for configuration data when PLL is the clock_source.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

◆ R_CGC_ClockStop()

ssp_err_t R_CGC_ClockStop (cgc_clock_t *clock_source*)

Stop the specified clock if it is active and not configured as the system clock.

Return values

SSP_SUCCESS	Clock stopped successfully.
SSP_ERR_CLOCK_ACTIVE	Current System clock source specified for stopping. This is not allowed.
SSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop MOCO when Oscillation stop is enabled.
SSP_ERR_NOT_STABILIZED	Clock not stabilized after starting. A finite stabilization time after starting the clock has to elapse before it can be stopped.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

◆ r_cgc_hoco_wait_control_set()

void r_cgc_hoco_wait_control_set (R_SYSTEM_Type * *p_system_reg*, uint8_t *hoco_wait*)

This function sets the HOCO wait time register.

Parameters

[in]	hoco_wait	HOCOWTCR HSTS setting
[in]	p_system_reg	pointer to system register structure

Return values

none	

◆ R_CGC_Init()

ssp_err_t R_CGC_Init (void)

Initialize the CGC API.

Configures the following for the clock generator module -If CGC_CFG_SUBCLOCK_AT_RESET_ENABLE is set to true:

- SubClock drive capacity (Compile time configurable: CGC_CFG_SUBCLOCK_DRIVE)
- Initial setting for the SubClock

THIS FUNCTION MUST BE EXECUTED ONCE AT STARTUP BEFORE ANY OF THE OTHER CGC FUNCTIONS CAN BE USED OR THE CLOCK SOURCE IS CHANGED FROM THE MOCO.

Return values

SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

SubClock will stop only if configurable setting is Enabled

◆ R_CGC_LCDClockCfg()

ssp_err_t R_CGC_LCDClockCfg (cgc_clock_t *clock*)

Configure the source for the segment LCDCLK.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.
SSP_ERR_INVALID_ARGUMENT	lcd_clock settings are invalid
SSP_ERR_UNSUPPORTED	lcd_clock configuration is not supported on this device

◆ R_CGC_LCDClockDisable()

ssp_err_t R_CGC_LCDClockDisable (void)

Disable the segment LCDCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.
SSP_ERR_UNSUPPORTED	lcd_clock is not supported on this device

◆ R_CGC_LCDClockEnable()

ssp_err_t R_CGC_LCDClockEnable (void)

Enable the segment LCDCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.
SSP_ERR_UNSUPPORTED	lcd_clock is not supported on this device

◆ r_cgc_operating_hw_modeset()

void r_cgc_operating_hw_modeset (R_SYSTEM_Type * *p_system_reg*, cgc_operating_modes_t *operating_mode*)

This function changes the operating power control mode.

Parameters

[in]	p_system_reg	pointer to system register structure
[in]	operating_mode	Operating power control mode

Enable writing to OPCCR and SOPCCR registers.

Wait for transition to complete.

Disable writing to OPCCR and SOPCCR registers.

The Sub-osc bit has to be cleared first.

Wait for transition to complete.

Set OPCCR.

Wait for transition to complete.

Set SOPCCR.

Wait for transition to complete.

Disable writing to OPCCR and SOPCCR registers.

◆ r_cgc_operating_mode_get()

cgc_operating_modes_t r_cgc_operating_mode_get (R_SYSTEM_Type * *p_system_reg*)

This function checks the MCU for High Speed Mode.

Parameters

[in]	p_system_reg	pointer to system register structure

Return values

operating_mode	current mode of operation read from OPCCR register

◆ R_CGC_OscStopDetect()

ssp_err_t R_CGC_OscStopDetect (void(*)(cgc_callback_args_t *p_args) *p_callback*, bool *enable*)

Enable or disable the oscillation stop detection for the main clock. The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.
SSP_ERR_ASSERTION	Null pointer passed for callback function when the second argument is "true".
SSP_ERR_ASSERTION	Cannot enable oscillator stop detect in sub-osc speed mode
SSP_ERR_ASSERTION	Invalid peripheral clock divisions for oscillator stop detect
SSP_ERR_INVALID_MODE	Invalid peripheral clock divider setting. Frequencies of peripherals should follow certain conditions.

add callback function to BSP

◆ R_CGC_OscStopStatusClear()

ssp_err_t R_CGC_OscStopStatusClear (void)

Clear the Oscillation Stop Detection Status register.

This register is not cleared automatically if the stopped clock is restarted. This function blocks for about 3 ICLK cycles until the status register is cleared.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.

◆ R_CGC_SDADCClockCfg()

ssp_err_t R_CGC_SDADCClockCfg (cgc_clock_t *clock*)

Configure the source for the SDADCCLK.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_UNSUPPORTED	sdadc_clock configuration is not supported on this device
SSP_ERR_INVALID_ARGUMENT	Invalid clock used

◆ R_CGC_SDADCClockDisable()

ssp_err_t R_CGC_SDADCClockDisable (void)

Disable the SDADCCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_UNSUPPORTED	sdadc_clock is not supported on this device

◆ R_CGC_SDADCClockEnable()

ssp_err_t R_CGC_SDADCClockEnable (void)

Enable the SDADCCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_UNSUPPORTED	sdadc_clock is not supported on this device

◆ R_CGC_SDRAMClockOutDisable()

ssp_err_t R_CGC_SDRAMClockOutDisable (void)

Disable the SDCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_UNSUPPORTED	sdram_clock is not supported on this device

◆ R_CGC_SDRAMClockOutEnable()

ssp_err_t R_CGC_SDRAMClockOutEnable (void)

Enable the SDCLK output.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_UNSUPPORTED	sdram_clock is not supported on this device

◆ R_CGC_SystemClockFreqGet()

ssp_err_t R_CGC_SystemClockFreqGet (cgc_system_clocks_t *clock*, uint32_t * *p_freq_hz*)

Return the requested internal clock frequency in Hz.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid clock specified.
SSP_ERR_ASSERTION	A NULL is passed for frequency data.

◆ R_CGC_SystemClockGet()

ssp_err_t R_CGC_SystemClockGet (cgc_clock_t * *clock_source*, cgc_system_clock_cfg_t * *p_set_clock_cfg*)

Return the current system clock source and configuration.

Return values

SSP_SUCCESS	Parameters returned successfully.
SSP_ERR_ASSERTION	A NULL is passed for configuration data.
SSP_ERR_ASSERTION	A NULL is passed for clock source.

◆ R_CGC_SystemClockSet()

ssp_err_t R_CGC_SystemClockSet (cgc_clock_t *clock_source*, cgc_system_clock_cfg_t const *const *p_clock_cfg*)

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD and FCLK.

THIS FUNCTION DOES NOT CHECK TO SEE IF THE OPERATING MODE SUPPORTS THE SPECIFIED CLOCK SOURCE AND DIVIDER VALUES. SETTING A CLOCK SOURCE AND DVIDER OUTSIDE THE RANGE SUPPORTED BY THE CURRENT OPERATING MODE WILL RESULT IN UNDEFINED OPERATION.

IF THE LOCO MOCO OR SUBCLOCK ARE CHOSEN AS THE SYSTEM CLOCK, THIS FUNCTION WILL SET THOSE AS THE SYSTEM CLOCK WITHOUT CHECKING FOR STABILIZATION. IT IS UP TO THE USER TO ENSURE THAT LOCO, MOCO OR SUBCLOCK ARE STABLE BEFORE USING THEM AS THE SYSTEM CLOCK.

Additionally this function sets the RAM and ROM wait states for the MCU. For the S7 MCU the ROMWT register controls ROM wait states. For the S3 MCU the MEMWAIT register controls ROM wait states.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_CLOCK_INACTIVE	The specified clock source is inactive.
SSP_ERR_ASSERTION	The p_clock_cfg parameter is NULL.
SSP_ERR_NOT_STABILIZED	The clock source has not stabilized
SSP_ERR_INVALID_ARGUMENT	Invalid argument used. ICLK is not set as the fastest clock.
SSP_ERR_INVALID_MODE	Peripheral divisions are not valid in sub-osc mode
SSP_ERR_INVALID_MODE	Oscillator stop detect not allowed in sub-osc mode

In order to correctly set the ROM and RAM wait state registers we need to know the current (S3A7 only) and requested iclk frequencies.

◆ R_CGC_SystickUpdate()

ssp_err_t R_CGC_SystickUpdate (uint32_t *period_count*, cgc_systick_period_units_t *units*)

Re-Configure the systick based on the provided period and current system clock frequency.

Parameters

[in]	period_count	The duration for the systick period.
[in]	units	The units for the provided period.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid period specified.
SSP_ERR_ABORTED	Attempt to update systick timer failed.

◆ R_CGC_USBClockCfg()

ssp_err_t R_CGC_USBClockCfg (cgc_usb_clock_div_t *divider*)

Configure the dividers for UCLK.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid usb_clock divider specified

◆ R_CGC_VersionGet()

ssp_err_t R_CGC_VersionGet (ssp_version_t *const *p_version*)

Return the driver version.

Return values

SSP_SUCCESS	Operation performed successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL..

5.1.5.10 CRC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the CRC Calculator (CRC). More...

Data Structures

struct	crc_instance_ctrl_t

Functions

ssp_err_t	R_CRC_Open (crc_ctrl_t *const p_api_ctrl, crc_cfg_t const *const p_cfg) Open the CRC driver module. More...
ssp_err_t	R_CRC_Close (crc_ctrl_t *const p_api_ctrl) Close the CRC module driver. More...
ssp_err_t	R_CRC_CalculatedValueGet (crc_ctrl_t *const p_api_ctrl, uint32_t *calculatedValue) Return the current calculated value. More...
ssp_err_t	R_CRC_SnoopEnable (crc_ctrl_t *const p_api_ctrl, uint32_t crc_seed) Enable snooping. More...
ssp_err_t	R_CRC_SnoopDisable (crc_ctrl_t *const p_api_ctrl) Disable snooping. More...
ssp_err_t	R_CRC_SnoopCfg (crc_ctrl_t *const p_api_ctrl, crc_snoop_cfg_t *const p_snoop_cfg) Configure the snoop channel and direction. More...
ssp_err_t	R_CRC_Calculate (crc_ctrl_t *const p_api_ctrl, void *inputBuffer, uint32_t length, uint32_t crc_seed, uint32_t *calculatedValue) Perform a CRC calculation on a block of 8-bit/32-bit(for 32-bit polynomial) data. More...
ssp_err_t	R_CRC_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...

Detailed Description

Driver for the CRC Calculator (CRC).

This module supports the CRC Calculator (CRC). It implements the following interface:

- CRC Interface

Function Documentation

◆ R_CRC_Calculate()

ssp_err_t R_CRC_Calculate (crc_ctrl_t *const *p_api_ctrl*, void * *inputBuffer*, uint32_t *length*, uint32_t *crc_seed*, uint32_t * *calculatedValue*)

Perform a CRC calculation on a block of 8-bit/32-bit(for 32-bit polynomial) data.

Implements crc_api_t::calculate

This function performs a CRC calculation on an array of 8-bit/32-bit(for 32-bit polynomial) values and returns an 8-bit/32-bit(for 32-bit polynomial) calculated value

Return values

SSP_SUCCESS	Calculation successful.
SSP_ERR_ASSERTION	Either p_ctrl, inputBuffer, or calculatedValue is NULL.
SSP_ERR_INVALID_ARGUMENT	length value is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.
SSP_ERR_IN_USE	CRC peripheral is currently in use by another instance of the driver.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Lock the peripheral during calculation

Set the bit order

Set CRC polynomial

Calculate CRC value for the input buffer

Release the hardware lock

◆ R_CRC_CalculatedValueGet()

ssp_err_t R_CRC_CalculatedValueGet (crc_ctrl_t *const *p_api_ctrl*, uint32_t * *calculatedValue*)

Return the current calculated value.

Implements crc_api_t::crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

Return values

SSP_SUCCESS	Return of calculated value successful.
SSP_ERR_ASSERTION	Either p_ctrl or calculatedValue is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

Based on the selected polynomial, return the calculated CRC value

◆ R_CRC_Close()

ssp_err_t R_CRC_Close (crc_ctrl_t *const *p_api_ctrl*)

Close the CRC module driver.

Implements crc_api_t::close

Return values

SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

Release the CRC Hardware Resource

Mark driver as closed

◆ R_CRC_Open()

ssp_err_t R_CRC_Open (crc_ctrl_t *const *p_api_ctrl*, crc_cfg_t const *const *p_cfg*)

Open the CRC driver module.

Implements crc_api_t::open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

Return values

SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Mark driver as initialized by setting the open value to the ASCII equivalent of "CRC"

Enable clocks to the CRC peripheral.

◆ R_CRC_SnoopCfg()

ssp_err_t R_CRC_SnoopCfg (crc_ctrl_t *const *p_api_ctrl*, crc_snoop_cfg_t *const *p_snoop_cfg*)

Configure the snoop channel and direction.

Implements crc_api_t::snoopCfg

The CRC calculator can operate on reads and writes over any of the first ten SCI channels. For example, if set to channel 0, transmit, every byte written out SCI channel 0 is also sent to the CRC calculator as if the value was explicitly written directly to the CRC calculator.

Return values

SSP_SUCCESS	Snoop configured successfully.
SSP_ERR_ASSERTION	- This is due to below conditions • Either p_ctrl or p_snoop_cfg is NULL • snoop_channel is greater than or equal to CRC_SNOOP_MAX_CHANNEL.
SSP_ERR_NOT_OPEN	The driver is not opened.

Set the bit order

Set CRC polynomial

Set CRC snoop channel and direction

◆ R_CRC_SnoopDisable()

ssp_err_t R_CRC_SnoopDisable (crc_ctrl_t *const *p_api_ctrl*)

Disable snooping.

Implements crc_api_t::snoopDisable

Return values

SSP_SUCCESS	Snoop disabled.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

Disable the snoop operation

◆ R_CRC_SnoopEnable()

ssp_err_t R_CRC_SnoopEnable (crc_ctrl_t *const *p_api_ctrl*, uint32_t *crc_seed*)

Enable snooping.

Implements crc_api_t::snoopEnable

Return values

SSP_SUCCESS	Snoop enabled.
SSP_ERR_ASSERTION	Either p_ctrl or crc_seed is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

Based on the selected polynomial, set the initial CRC seed value

Enable the snoop operation

◆ R_CRC_VersionGet()

ssp_err_t R_CRC_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Implements crc_api_t::versionGet

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

crc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CRC

#include <r_crc.h>

Data Fields

R_CRC_Type *	p_reg Pointer to register base address.
uint32_t	open Whether or not channel is open.
crc_polynomial_t	polynomial CRC Generating Polynomial Switching (GPS).
crc_bit_order_t	bit_order CRC Calculation Switching (LMS).
bool	fifo_mode FIFO Mode selection for sci_uart in CRC snoop operation.

Detailed Description

Driver instance control structure.

The documentation for this struct was generated from the following file:

- r_crc.h

5.1.5.11 CTSU v2

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Capacitive Touch Sensing Unit (CTSU). More...

Data Structures

struct	ctsu_ctsuwr_t
struct	ctsu_self_buf_t
struct	ctsu_mutual_buf_t
struct	ctsu_correction_info_t
struct	ctsu_instance_ctrl_t

Enumerations

enum	ctsu_state_t { CTSU_STATE_INIT, CTSU_STATE_IDLE, CTSU_STATE_SCANNING, CTSU_STATE_SCANNED }
enum	ctsu_tuning_t { CTSU_TUNING_INCOMPLETE, CTSU_TUNING_COMPLETE }
enum	ctsu_correction_status_t { CTSU_CORRECTION_INIT, CTSU_CORRECTION_RUN, CTSU_CORRECTION_COMPLETE, CTSU_CORRECTION_ERROR }

Functions

ssp_err_t	R_CTSU_Open (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg) Opens and configures the CTSU driver module. Implements ctsu_api_t::open. More...
ssp_err_t	R_CTSU_ScanStart (ctsu_ctrl_t *const p_ctrl) This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart. More...
ssp_err_t	R_CTSU_DataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data) This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet. More...
ssp_err_t	R_CTSU_CallbackSet (ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)

ssp_err_t	R_CTSU_Close (ctsu_ctrl_t *const p_ctrl) Disables specified CTSU control block. Implements ctsu_api_t::close. More...
ssp_err_t	R_CTSU_VersionGet (ssp_version_t *const p_version) Return CTSU HAL driver version. Implements ctsu_api_t::versionGet. More...
ssp_err_t	R_CTSU_Diagnosis (ctsu_ctrl_t *const p_ctrl) Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis. More...

Detailed Description

Driver for the Capacitive Touch Sensing Unit (CTSU).

Enumeration Type Documentation

◆ ctsu_correction_status_t

enum ctsu_correction_status_t	
CTSU Correction status	
Enumerator	
CTSU_CORRECTION_INIT	Correction initial status.
CTSU_CORRECTION_RUN	Correction scan running.
CTSU_CORRECTION_COMPLETE	Correction complete.
CTSU_CORRECTION_ERROR	Correction error.

◆ ctsu_state_t

enum ctsu_state_t	
CTSU run state	
Enumerator	
CTSU_STATE_INIT	Not open.
CTSU_STATE_IDLE	Opened.
CTSU_STATE_SCANNING	Scanning now.
CTSU_STATE_SCANNED	Scan end.

◆ ctsu_tuning_t

enum ctsu_tuning_t	
CTSU Initial offset tuning status	
Enumerator	
CTSU_TUNING_INCOMPLETE	Initial offset tuning incomplete.
CTSU_TUNING_COMPLETE	Initial offset tuning complete.

Function Documentation

◆ R_CTSU_CallbackSet()

ssp_err_t R_CTSU_CallbackSet (ctsu_ctrl_t *const *p_api_ctrl*, void(*)(ctsu_callback_args_t *) *p_callback*, void const *const *p_context*, ctsu_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements ctsu_api_t::callbackSet

Return values

SSP_SUCCESS	Callback updated successfully.
SSP_ERR_ASSERTION	A required pointer is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_CTSU_Close()

ssp_err_t R_CTSU_Close (ctsu_ctrl_t *const *p_ctrl*)

Disables specified CTSU control block. Implements ctsu_api_t::close.

Return values

SSP_SUCCESS	CTSU successfully configured.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.

Stops peripheral and clears any internal state to allow driver to be reconfigured.

◆ R_CTSU_DataGet()

ssp_err_t R_CTSU_DataGet (ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_data*)

This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet.

Return values

SSP_SUCCESS	CTSU successfully configured.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.
SSP_ERR_CTSU_SCANNING	Scanning this instance.
SSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.
SSP_ERR_CTSU_DIAG_NOT_YET	Diagnosis of data collected no yet.

◆ R_CTSU_Diagnosis()

ssp_err_t R_CTSU_Diagnosis (ctsu_ctrl_t *const *p_ctrl*)

Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis.

Return values

SSP_SUCCESS	CTSU successfully configured.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.
SSP_ERR_NOT_ENABLED	Diagnosis is not enabled in S7 Series
SSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.
SSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE	Diagnosis of LDO over voltage failed.
SSP_ERR_CTSU_DIAG_CCO_HIGH	Diagnosis of CCO into 19.2uA failed.
SSP_ERR_CTSU_DIAG_CCO_LOW	Diagnosis of CCO into 2.4uA failed.
SSP_ERR_CTSU_DIAG_SSCG	Diagnosis of SSCG frequency failed.
SSP_ERR_CTSU_DIAG_DAC	Diagnosis of non-touch count value failed.

◆ R_CTSU_Open()

ssp_err_t R_CTSU_Open (ctsu_ctrl_t *const *p_ctrl*, ctsu_cfg_t const *const *p_cfg*)

Opens and configures the CTSU driver module. Implements ctsu_api_t::open.

Return values

SSP_SUCCESS	CTSU successfully configured.
SSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
SSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
SSP_ERR_INVALID_ARGUMENT	Configuration parameter error.
SSP_ERR_IN_USE	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- cgc_api_t::systemClockFreqGet
- transfer_api_t::open

Note

In the first Open, measurement for correction works, and it takes several tens of milliseconds.

◆ R_CTSU_ScanStart()

ssp_err_t R_CTSU_ScanStart (ctsu_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart.

Return values

SSP_SUCCESS	CTSU successfully configured.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Module is not open.
SSP_ERR_CTSU_SCANNING	Scanning this instance or other.
SSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.

< CTSU_STRT

◆ R_CTSU_VersionGet()

ssp_err_t R_CTSU_VersionGet (ssp_version_t *const *p_version*)

Return CTSU HAL driver version. Implements ctsu_api_t::versionGet.

Return values

SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter

ctsu_ctsuwr_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CTSU v2

#include <r_ctsuv2.h>

Data Fields

uint16_t	ctsussc Copy from (ssdiv << 8) by Open API.

uint16_t	ctsuso0 Copy from ((snum << 10) | so) by Open API.
uint16_t	ctsuso1 Copy from (sdpa << 8) by Open API. ICOG and RICOA is set recommend value.

Detailed Description

CTSUWR write register value

The documentation for this struct was generated from the following file:

- r_ctsuv2.h

ctsu_self_buf_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CTSU v2

#include <r_ctsuv2.h>

Data Fields

uint16_t	sen Sensor counter data.
uint16_t	ref Reference counter data (Not used)

Detailed Description

Scan buffer data formats (Self)

The documentation for this struct was generated from the following file:

- r_ctsuv2.h

ctsu_mutual_buf_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CTSU v2

#include <r_ctsuv2.h>

Data Fields

uint16_t	pri_sen Primary sensor data.
uint16_t	pri_ref Primary reference data (Not used)
uint16_t	snd_sen Secondary sensor data.
uint16_t	snd_ref Secondary reference data (Not used)

Detailed Description

Scan buffer data formats (Mutual)

The documentation for this struct was generated from the following file:

- r_ctsuv2.h

ctsu_correction_info_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » CTSU v2

#include <r_ctsuv2.h>

Data Fields

ctsu_correction_status_t	status

	Correction status.
ctsu_ctsuwr_t	ctsuwr Correction scan parameter.
volatile ctsu_self_buf_t	scanbuf Correction scan buffer.
uint16_t	first_val 1st correction value
uint16_t	second_val 2nd correction value
uint32_t	first_coefficient 1st correction coefficient
uint32_t	second_coefficient 2nd correction coefficient
uint32_t	ctsu_clock CTSU clock [MHz].

Detailed Description

Correction information

The documentation for this struct was generated from the following file:

- r_ctsuv2.h

ctsu_instance_ctrl_t Struct Reference
Renesas Synergy Software Package Reference » HAL Layer » CTSU v2

#include <r_ctsuv2.h>

Data Fields

uint32_t	open Whether or not driver is open.
ctsu_state_t	state CTSU run state.
ctsu_md_t	md CTSU Measurement Mode Select(copy to cfg)
ctsu_tuning_t	tuning CTSU Initial offset tuning status.
uint16_t	num_elements Number of elements to scan.
uint16_t	wr_index Word index into ctsuwr register array.
uint16_t	rd_index Word index into scan data buffer.
uint8_t *	p_tuning_count Pointer to tuning count of each element. g_ctsu_tuning_count[] is set by Open API.
int32_t *	p_tuning_diff Pointer to difference from base value of each element. g_ctsu_tuning_diff[] is set by Open API.
uint16_t	average CTSU Moving average counter.

uint16_t	num_moving_average Copy from config by Open API.
uint8_t	ctsucr1 Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW, and PON is set by HAL driver.
ctsu_ctsuwr_t *	p_ctsuwr CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API.
ctsu_self_buf_t *	p_self_raw Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API.
uint16_t *	p_self_data Pointer to Self moving average data. g_ctsu_self_data[] is set by Open API.
ctsu_mutual_buf_t *	p_mutual_raw Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open API.
uint16_t *	p_mutual_pri_data Pointer to Mutual primary moving average data. g_ctsu_mutual_pri_data[] is set by Open API.
uint16_t *	p_mutual_snd_data Pointer to Mutual secondary moving average data. g_ctsu_mutual_snd_data[] is set by Open API.
ctsu_correction_info_t *	p_correction_info Pointer to correction info.
uint8_t	ctsuchac0 TS00-TS07 enable mask.
uint8_t	ctsuchac1

	TS08-TS15 enable mask.
uint8_t	ctsuchac2 TS16-TS23 enable mask.
uint8_t	ctsuchac3 TS24-TS31 enable mask.
uint8_t	ctsuchac4 TS32-TS39 enable mask.
uint8_t	ctsuchtrc0 TS00-TS07 mutual-tx mask.
uint8_t	ctsuchtrc1 TS08-TS15 mutual-tx mask.
uint8_t	ctsuchtrc2 TS16-TS23 mutual-tx mask.
uint8_t	ctsuchtrc3 TS24-TS31 mutual-tx mask.
uint8_t	ctsuchtrc4 TS32-TS39 mutual-tx mask.
ctsu_cfg_t const *	p_ctsu_cfg Pointer to initial configurations.
IRQn_Type	write_irq Copy from config by Open API. CTSU_CTSUWR interrupt vector.
IRQn_Type	read_irq Copy from config by Open API. CTSU_CTSURD interrupt vector.

IRQn_Type	end_irq Copy from config by Open API. CTSU_CTSUFN interrupt vector.
void(*	p_callback)(ctsu_callback_args_t *) Callback provided when a CTSUFN occurs.
ctsu_callback_args_t *	p_callback_memory Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
void const *	p_context Placeholder for user data.
R_CTSU_Type *	p_reg Pointer to base register address.
uint16_t	tuning_self_target_value Target self value for initial offset tuning.
uint16_t	tuning_mutual_target_value Target mutual value for initial offset tuning.

Detailed Description

CTSU private control block. DO NOT MODIFY. Initialization occurs when R_CTSU_Open() is called.

The documentation for this struct was generated from the following file:

- r_ctsuv2.h

5.1.5.12 DAC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the 12-Bit D/C Converter (DAC12). More...

Data Structures

struct	dac_instance_ctrl_t
struct	dac_extended_cfg_t

Functions

ssp_err_t	R_DAC_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg) Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first. More...
ssp_err_t	R_DAC_Close (dac_ctrl_t *p_api_ctrl) Stop the D/A conversion, stop output, and close the DAC channel. More...
ssp_err_t	R_DAC_Write (dac_ctrl_t *p_api_ctrl, dac_size_t value) Write data to the D/A converter and enable the output if it has not been enabled. More...
ssp_err_t	R_DAC_Start (dac_ctrl_t *p_api_ctrl) Start the D/A conversion output if it has not been started. More...
ssp_err_t	R_DAC_Stop (dac_ctrl_t *p_api_ctrl) Stop the D/A conversion and disable the output signal. More...
ssp_err_t	R_DAC_VersionGet (ssp_version_t *p_version) Get version and store it in provided pointer p_version. More...
ssp_err_t	R_DAC_InfoGet (dac_info_t *const p_info) Get information about DAC Resolution and store it in provided pointer p_info. More...

Detailed Description

Driver for the 12-Bit D/C Converter (DAC12).

Summary

This module implements the following interface: DAC Interface.

Name of module used by error logger macro

Function Documentation

◆ R_DAC_Close()

ssp_err_t R_DAC_Close (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

Return values

SSP_SUCCESS	The channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Validate that the channel is opened.

Stop the channel

Update the channel state information.

Unlock the DAC Hardware Resource

Power down the DAC device.

Unlock the DAC Hardware Resource

◆ R_DAC_InfoGet()

ssp_err_t R_DAC_InfoGet (dac_info_t *const *p_info*)

Get information about DAC Resolution and store it in provided pointer p_info.

Return values

SSP_SUCCESS	Value of DAC resolution written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_info parameter was null.

Assigning DAC bit resolution as 12bit.

◆ R_DAC_Open()

ssp_err_t R_DAC_Open (dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg*)

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

Return values

SSP_SUCCESS	The channel was successfully opened.
SSP_ERR_ASSERTION	One or both of the following parameters may be NULL: p_api_ctrl or p_cfg Channel ID requested in p_cfg may not available on the device selected in r_bsp_config.h data_format value in p_cfg is out of range. ad_da_synchronized value in p_cfg is out of range.
SSP_ERR_IN_USE	DAC resource is locked.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Validate the input parameter.

Make sure the peripheral exists.

Lock the DAC Hardware Resource.

Power on the DAC device.

Stop the channel.

Configure data format: left or right justified.

Configure D/A-A/D Synchronous Start Control Register(DAADSCR).

Set output amplifier configuration for the channel.

Set the reference voltage.

If charge pump has supported by MCU, configures the register(DAPC)

Initialize the channel state information.

All done. Return.

◆ R_DAC_Start()

ssp_err_t R_DAC_Start (dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output if it has not been started.

Return values

SSP_SUCCESS	The channel is started successfully.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Validate that the channel is opened.

Enable the output.

Update the internal state.

◆ R_DAC_Stop()

ssp_err_t R_DAC_Stop (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

Return values

SSP_SUCCESS	The control is successfully stopped.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Validate that the channel is opened.

Disable the output.

Mark the internal state.

◆ R_DAC_VersionGet()

ssp_err_t R_DAC_VersionGet (ssp_version_t * *p_version*)

Get version and store it in provided pointer p_version.

Return values

SSP_SUCCESS	Successfully retrieved version information.
SSP_ERR_ASSERTION	p_version is NULL.

◆ R_DAC_Write()

ssp_err_t R_DAC_Write (dac_ctrl_t * *p_api_ctrl*, dac_size_t *value*)

Write data to the D/A converter and enable the output if it has not been enabled.

Return values

SSP_SUCCESS	Data is successfully written to the D/A Converter.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Note

Write function automatically starts the D/A conversion after data is successfully written to the channel.

Validate that the channel is opened.

Write the value to D/A converter.

Start the converter if it has been idle.

Start the channel

dac_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DAC

#include <r_dac.h>

Data Fields

void *	p_reg Pointer to DAC base register.
uint8_t	channel ID associated with this DAC channel.
uint8_t	channel_started DAC operation on channel started.
uint8_t	channel_opened DAC channel open.

Detailed Description

DAC instance control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_dac.h

dac_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DAC

#include <r_dac.h>

Data Fields

bool	enable_charge_pump Enable DAC charge pump.

Detailed Description

DAC extended configuration

The documentation for this struct was generated from the following file:

- r_dac.h

5.1.5.13 DAC8

Renesas Synergy Software Package Reference » HAL Layer

Driver for the 8-Bit D/C Converter (DAC8). More...

Data Structures

struct	dac8_instance_ctrl_t
struct	dac8_extended_cfg_t

Enumerations

enum	dac8_mode_t { DAC8_MODE_NORMAL, DAC8_MODE_REAL_TIME }

Functions

ssp_err_t	R_DAC8_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg) Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first. More...
ssp_err_t	R_DAC8_Close (dac_ctrl_t *p_api_ctrl) Stop the D/A conversion, stop output, and close the DAC channel. More...
ssp_err_t	R_DAC8_Write (dac_ctrl_t *p_api_ctrl, dac_size_t value) Write data to the D/A converter and enable the output if it has not been enabled. More...
ssp_err_t	R_DAC8_Start (dac_ctrl_t *p_api_ctrl) Start the D/A conversion output. More...
ssp_err_t	R_DAC8_Stop (dac_ctrl_t *p_api_ctrl) Stop the D/A conversion and disable the output signal. More...
ssp_err_t	R_DAC8_VersionGet (ssp_version_t *p_version) Get version and store it in provided pointer p_version. More...
ssp_err_t	R_DAC8_InfoGet (dac_info_t *const p_info) Get information about DAC Resolution and store it in provided pointer p_info. More...

Detailed Description

Driver for the 8-Bit D/C Converter (DAC8).

Summary

This module implements the following interface: DAC Interface.

Name of module used by error logger macro

Enumeration Type Documentation

◆ dac8_mode_t

enum dac8_mode_t	
DAC8 mode	
Enumerator	
DAC8_MODE_NORMAL	DAC Normal mode.
DAC8_MODE_REAL_TIME	DAC Real-time (event link) mode.

Function Documentation

◆ R_DAC8_Close()

ssp_err_t R_DAC8_Close (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

Return values

SSP_SUCCESS	The channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Stop the channel

Update the channel state information.

Update DAC Hardware Resource information.

Unlock the DAC Hardware Resource

◆ R_DAC8_InfoGet()

ssp_err_t R_DAC8_InfoGet (dac_info_t *const *p_info*)

Get information about DAC Resolution and store it in provided pointer p_info.

Return values

SSP_SUCCESS	Value of DAC resolution written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_info parameter was null.

Update DAC resolution as 8-bit

◆ R_DAC8_Open()

ssp_err_t R_DAC8_Open (dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg*)

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

Return values

SSP_SUCCESS	The channel was successfully opened.
SSP_ERR_ASSERTION	One or both of the following parameters may be NULL: p_api_ctrl or p_cfg
SSP_ERR_IN_USE	DAC8 resource is locked.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	An invalid channel was requested.
SSP_ERR_UNSUPPORTED	Output amplifier is not supported. Real time mode is not supported. Charge pump is not supported. A/D - D/A synchronization is not supported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Validate the input parameter.

Make sure the peripheral exists.

Lock the DAC Hardware Resource.

Power on the DAC device.

Stop the channel.

Set defaults.

Configure the charge pump.

Configure the DAC mode.

Configure DA/AD mode.

Initialize the channel state information.

All done. Return.

◆ R_DAC8_Start()

ssp_err_t R_DAC8_Start (dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output.

Return values

SSP_SUCCESS	The channel is started successfully.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Enable the output.

Update the internal state.

◆ R_DAC8_Stop()

ssp_err_t R_DAC8_Stop (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

Return values

SSP_SUCCESS	The control is successfully stopped.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

Disable the output.

Mark the internal state.

◆ R_DAC8_VersionGet()

ssp_err_t R_DAC8_VersionGet (ssp_version_t * *p_version*)

Get version and store it in provided pointer p_version.

Return values

SSP_SUCCESS	Successfully retrieved version information.
SSP_ERR_ASSERTION	p_version is NULL.

Return the version number

◆ R_DAC8_Write()

ssp_err_t R_DAC8_Write (dac_ctrl_t * *p_api_ctrl*, dac_size_t *value*)

Write data to the D/A converter and enable the output if it has not been enabled.

Return values

SSP_SUCCESS	Data is successfully written to the D/A Converter.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.
SSP_ERR_OVERFLOW	Data overflow when data value exceeds 8-bit limit.

Note

Write function automatically starts the D/A conversion after data is successfully written to the channel.

Handle data format.

Convert to 8 bits.

Write the value to D/A converter.

Start the converter if it has been idle.

Start the channel

dac8_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DAC8

#include <r_dac8.h>

Data Fields

void *	p_reg Pointer to DAC base register.
uint8_t	channel ID associated with this DAC channel.
uint8_t	channel_started DAC operation on channel started.

uint32_t	channel_opened DAC channel open.
dac_data_format_t	data_format DAC data format.

Detailed Description

DAC8 instance control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_dac8.h

dac8_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DAC8

#include <r_dac8.h>

Data Fields

bool	enable_charge_pump Enable DAC charge pump.
dac8_mode_t	dac_mode DAC mode.

Detailed Description

DAC8 extended configuration

The documentation for this struct was generated from the following file:

- r_dac8.h

5.1.5.14 DMAC

Renesas Synergy Software Package Reference » HAL Layer

DMA Controller (DMAC). More...

Data Structures

struct	dmac_instance_ctrl_t
struct	transfer_on_dmac_cfg_t

Macros

#define	DMAC_REPEAT_BLOCK_MAX_LENGTH (0x400)
#define	DMAC_NORMAL_MAX_LENGTH (0xFFFF)
#define	DUMMY_ADDRESS ((void *) 0x40005500)

Functions

ssp_err_t	R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) Initialize transfer and enables transfer in ICU. Implements transfer_api_t::open. More...
ssp_err_t	R_DMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers) Reset transfer source, destination, and number of transfers. More...
ssp_err_t	R_DMAC_Start (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) Start transfer. Implements transfer_api_t::start. More...
ssp_err_t	R_DMAC_Stop (transfer_ctrl_t *const p_api_ctrl) Stop transfer. Implements transfer_api_t::stop. More...
ssp_err_t	R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl) Enable transfer. Implements transfer_api_t::enable. More...
ssp_err_t	R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl)

	Disable transfer. Implements transfer_api_t::disable. More...
ssp_err_t	R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info) Set driver specific information in provided pointer. Implements transfer_api_t::infoGet. More...
ssp_err_t	R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl) Disable transfer and clean up internal data. Implements transfer_api_t::close. More...
ssp_err_t	R_DMAC_VersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. More...
ssp_err_t	R_DMAC_BlockReset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const length, transfer_size_t size, uint16_t const num_transfers) Reset transfer source, destination, length and number of transfers for block transfer. More...
ssp_err_t	R_DMAC_Stop_ActivationRequest (transfer_ctrl_t *const p_api_ctrl) Clears the DMA activation request with a DMA dummy transfer as per flowchart in the hardware manual. Implements transfer_api_t::Stop_ActivationRequest. This function to be used only in scenario when a DMA activation request source might occur in the next request after a DMA transfer completes. If this happens, the DMA transfer starts and the DMA activation request is held in DMAC. More...

Detailed Description

DMA Controller (DMAC).

Summary

Extends the Transfer Interface.

The Direct Memory Access (DMA) Controller allows data transfers to occur in place of or in addition to any interrupt. It also supports data transfers using software start.

Note

The transfer length is limited to 1024 (10 bits) in TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT.

This driver supports only TRANSFER_IRQ_END from transfer_irq_t.

Macro Definition Documentation

◆ DMAC_NORMAL_MAX_LENGTH

#define DMAC_NORMAL_MAX_LENGTH (0xFFFF)
Length limited to 65535 transfers for normal mode

◆ DMAC_REPEAT_BLOCK_MAX_LENGTH

#define DMAC_REPEAT_BLOCK_MAX_LENGTH (0x400)
Length limited to 1024 transfers for repeat and block mode

◆ DUMMY_ADDRESS

#define DUMMY_ADDRESS ((void *) 0x40005500)
Reserved memory area for dummy write transfer as per Hardware user manual

Function Documentation

◆ R_DMAC_BlockReset()

ssp_err_t R_DMAC_BlockReset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *length*, transfer_size_t *size*, uint16_t const *num_transfers*)

Reset transfer source, destination, length and number of transfers for block transfer.

Return values

SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.
SSP_ERR_NOT_ENABLED	DMAC is not enabled. A valid source and destination must be provided in either open() or blockReset().

Enables transfers on this activation source.

◆ R_DMAC_Close()

ssp_err_t R_DMAC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer and clean up internal data. Implements transfer_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.

Disable DMAC transfers, disable DMAC interrupts, and remove DMAC trigger from ICU register.

Clear ID so control block can be reused.

Release BSP hardware lock on this channel

◆ R_DMAC_Disable()

ssp_err_t R_DMAC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer. Implements transfer_api_t::disable.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

Disable transfer.

◆ R_DMAC_Enable()

ssp_err_t R_DMAC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfer. Implements transfer_api_t::enable.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

Enable transfer.

◆ R_DMAC_InfoGet()

ssp_err_t R_DMAC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info*)

Set driver specific information in provided pointer. Implements transfer_api_t::infoGet.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_ASSERTION	An input parameter is invalid.

If a transfer is active, store it in p_in_progress.

Store maximum transfer length.

Store remaining transfer length.

◆ R_DMAC_Open()

ssp_err_t R_DMAC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)

Initialize transfer and enables transfer in ICU. Implements transfer_api_t::open.

Return values

SSP_SUCCESS	Successful open. Transfer is configured and will start when trigger occurs.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_ENABLED	Auto-enable was requested, but enable failed.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
SSP_ERR_INVALID_SIZE	Invalid offset value.
SSP_ERR_IN_USE	The BSP hardware lock for the DMAC is not available.

Returns

See Common Error Codes for other possible return codes.This function calls

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Get the IRQ vectors, event info and set the NVIC priority for dmac.

Acquire BSP hardware lock for channel used.

Configure the DMAC according to the flowchart "Activating the DMAC" in chapter 16.3.7 of hardware manual NoSecurity_r01uh0488ej0040_sc32.pdf.

Set the offset value in offset addressing mode.

Configure DMA transfer and sources

Note

Transfer escape interrupts not supported.

Update internal variables.

Mark driver as open by initializing "DMAC" in its ASCII equivalent.

If auto_enable is true, enable transfer and ELC events if software start is used.

◆ R_DMAC_Reset()

ssp_err_t R_DMAC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers.

Return values

SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_ENABLED	DMAC is not enabled. A valid source and destination must be provided in either open() or reset().
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.

Enables transfers on this activation source.

◆ R_DMAC_Start()

ssp_err_t R_DMAC_Start (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

Start transfer. Implements transfer_api_t::start.

Return values

SSP_SUCCESS	Transfer started written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	Handle was not configured for software activation.

Set autoclear bit and software start bit.

◆ R_DMAC_Stop()

ssp_err_t R_DMAC_Stop (transfer_ctrl_t *const *p_api_ctrl*)

Stop transfer. Implements transfer_api_t::stop.

Return values

SSP_SUCCESS	Transfer stopped written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

Reset auto clear bit and clear software transfer request.

◆ R_DMAC_Stop_ActivationRequest()

ssp_err_t R_DMAC_Stop_ActivationRequest (transfer_ctrl_t *const *p_api_ctrl*)

Clears the DMA activation request with a DMA dummy transfer as per flowchart in the hardware manual. Implements transfer_api_t::Stop_ActivationRequest. This function to be used only in scenario when a DMA activation request source might occur in the next request after a DMA transfer completes. If this happens, the DMA transfer starts and the DMA activation request is held in DMAC.

Return values

SSP_SUCCESS	Successful transfer.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

Clear the DMA activation request with a DMA dummy transfer as per flowchart in the hardware manual.

Disable DMAC transfer.

Disable the IRQ pin as a DMACm request source

Set the DMAC transfer size

Set source and destination address to 4000_5500 as per hardware manual.

Set number of transfer operations

Disable DMAC transfer.

Wait for the DMAC transfer end

◆ R_DMAC_VersionGet()

ssp_err_t R_DMAC_VersionGet (ssp_version_t *const *p_version*)
Set driver version based on compile time macros. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — An input parameter is invalid.

dmac_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DMAC

#include <r_dmac.h>

Data Fields

uint32_t	id Driver ID.
elc_event_t	trigger Transfer activation event. Matches event returned by transfer_api_t::infoGet.
IRQn_Type	irq Transfer activation IRQ.
uint8_t	channel Channel number.
uint8_t	ir_flag_stat Status of IR bit in DELSR register.
int32_t	offset_byte Value of offset in bytes for offset addressing mode.

void(*	p_callback)(transfer_callback_args_t *cb_data)
void const *	p_context
void *	p_reg

Detailed Description

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

Field Documentation

◆ p_callback

void(* dmac_instance_ctrl_t::p_callback) (transfer_callback_args_t *cb_data)
Callback for transfer end interrupt.

◆ p_context

void const* dmac_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

◆ p_reg

void* dmac_instance_ctrl_t::p_reg
Pointer to base register.

The documentation for this struct was generated from the following file:

- r_dmac.h

transfer_on_dmac_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DMAC

#include <r_dmac.h>

Data Fields

uint8_t	channel Channel number, does not apply to all HAL drivers.
int32_t	offset_byte Value of offset in bytes for offset addressing mode.

Detailed Description

DMAC transfer configuration extension. This extension is required.

The documentation for this struct was generated from the following file:

- r_dmac.h

5.1.5.15 DOC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Data Operation Circuit (DOC). More...

Data Structures

struct	doc_instance_ctrl_t

Functions

ssp_err_t	R_DOC_Open (doc_ctrl_t *const p_api_ctrl, doc_cfg_t const *const p_cfg) Configure the Data Operation Circuit (DOC) in comparison, addition or subtraction mode. Implements doc_api_t::open. More...
ssp_err_t	R_DOC_Close (doc_ctrl_t *const p_api_ctrl) Close the module driver. Enable module stop mode. Implements doc_api_t::close. More...
ssp_err_t	R_DOC_StatusGet (doc_ctrl_t *const p_api_ctrl, doc_status_t *p_status) Return the comparison/addition/subtraction status. Implements doc_api_t::statusGet. More...

ssp_err_t	R_DOC_StatusClear (doc_ctrl_t *const p_api_ctrl) Clear the DOPCF status flag at the hardware layer. This flag indicates the result of a DOC operation. Implements doc_api_t::statusClear. More...
ssp_err_t	R_DOC_Write (doc_ctrl_t *const p_api_ctrl, doc_data_t *const p_data) Write to the DODIR and DODSR registers. Implements doc_api_t::write,. More...
ssp_err_t	R_DOC_InputRegisterWrite (doc_ctrl_t *const p_api_ctrl, doc_size_t data) Write to the DODIR register. Implements doc_api_t::inputRegisterWrite,. More...
ssp_err_t	R_DOC_VersionGet (ssp_version_t *const p_version) Return DOC HAL driver version. Implements doc_api_t::versionGet. More...

Detailed Description

Driver for the Data Operation Circuit (DOC).

Summary

This module implements the DOC Interface using the Data Operation Circuit (DOC).

Function Documentation

◆ R_DOC_Close()

ssp_err_t R_DOC_Close (doc_ctrl_t *const *p_api_ctrl*)

Close the module driver. Enable module stop mode. Implements doc_api_t::close.

To close the DOC it must have been opened via the open API. When opened a control structure of type doc_ctrl_t is passed to the open API. The same control structure must be passed to the close API.

Return values

SSP_SUCCESS	Module successfully closed.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::eventInfoGet

Note

This function is reentrant.
This function will disable the DOC interrupt in the NVIC.

Validate the parameter and check if the module is initialized

Disable the IRQ in the NVIC in case it has been left enabled.

Clear DOPCF in DOCR

Mark driver as closed.

Unlock the DOC Hardware Resource

◆ R_DOC_InputRegisterWrite()

ssp_err_t R_DOC_InputRegisterWrite (doc_ctrl_t *const *p_api_ctrl*, doc_size_t *data*)

Write to the DODIR register. Implements doc_api_t::inputRegisterWrite,.

Writes to the DODIR register only.

Return values

SSP_SUCCESS	Value successfully written to the DODIR register.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.

Note

This function is reentrant.

Validate the parameter and check if the module is intialized

Writes the user supplied data to the DODIR register for data operation in Comparison, Addition and subtraction modes

◆ R_DOC_Open()

ssp_err_t R_DOC_Open (doc_ctrl_t *const *p_api_ctrl*, doc_cfg_t const *const *p_cfg*)

Configure the Data Operation Circuit (DOC) in comparison, addition or subtraction mode. Implements doc_api_t::open.

If callback is not NULL in the configuration structure the DOC IRQ will be enabled.

Return values

SSP_SUCCESS	DOC successfully configured.
SSP_ERR_IN_USE	Module already open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_INVALID_ARGUMENT	ISR is not enabled. Enable the ISR in bsp_irq_cfg.h.
SSP_ERR_HW_LOCKED	DOC resource is locked.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is reentrant.

Validate the parameters and check if the module is initialized

Make sure the DOC exists on this part.

Lock the DOC Hardware Resource

Configure the DOC via DOCR register.

If callback parameter is not NULL configure the IRQ

Mark driver as open by initializing it to "DOCO" in its ASCII equivalent.

◆ R_DOC_StatusClear()

ssp_err_t R_DOC_StatusClear (doc_ctrl_t *const *p_api_ctrl*)

Clear the DOPCF status flag at the hardware layer. This flag indicates the result of a DOC operation. Implements doc_api_t::statusClear.

Return values

SSP_SUCCESS	Interrupt successfully cleared.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

Note

This function is reentrant.

Validate the parameter and check if the module is initialized

Clear the hardware status flag

◆ R_DOC_StatusGet()

ssp_err_t R_DOC_StatusGet (doc_ctrl_t *const *p_api_ctrl*, doc_status_t * *p_status*)

Return the comparison/addition/subtraction status. Implements doc_api_t::statusGet.

The status is read from the Data Operation Circuit Flag (DOPCF).

Return values

SSP_SUCCESS	Status successfully read.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.

Note

This function is reentrant.

Validate the parameters and check if the module is intialized

Read the comparison or addition or subtraction status from the register and store in the user supplied location

◆ R_DOC_VersionGet()

ssp_err_t R_DOC_VersionGet (ssp_version_t *const *p_version*)

Return DOC HAL driver version. Implements doc_api_t::versionGet.

Return values

SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

Note

This function is reentrant.

Validate the parameter

Store the version id from version data structure to the user supplied location

◆ R_DOC_Write()

ssp_err_t R_DOC_Write (doc_ctrl_t *const *p_api_ctrl*, doc_data_t *const *p_data*)

Write to the DODIR and DODSR registers. Implements doc_api_t::write,.

In comparison mode the 16-bit reference data is written to the DODSR register and the data for the comparison written to the DODIR. In addition mode the initial data is written to the DODSR and the data to be added to the DODIR. In subtraction mode the initial data is written to the DODSR and the data to be subtracted to the DODIR.

When changing both the DODSR and DODIR the DODSR should be updated first.

Return values

SSP_SUCCESS	Values successfully written to the registers.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.

Note

This function is reentrant.

Validate the parameters and check if the module is initialized

Writes the user supplied data to the DODIR(DOC Data Input Register) and DODSR(DOC Data Setting Register) registers

doc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DOC

#include <r_doc.h>

Data Fields

uint32_t	open Used by driver to check if the control structure is valid.
void(*	p_callback)(doc_callback_args_t *p_args)
doc_event_t	event The event DOC is configured for. Passed in ISR callback.
void const *	p_context
void *	p_reg Base register.

Detailed Description

DOC instance control block. Do not initialize. Initialization occurs when the doc_api_t::open function is called.

Field Documentation

◆ p_callback

void(* doc_instance_ctrl_t::p_callback) (doc_callback_args_t *p_args)
Callback provided when a DOC ISR occurs. NULL indicates no CPU interrupt.

◆ p_context

void const* doc_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in doc_callback_args_t.

The documentation for this struct was generated from the following file:

- r_doc.h

5.1.5.16 DTC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Data Transfer Controller (DTC). More...

Data Structures

struct	dtc_instance_ctrl_t
struct	dtc_reg_t

Macros

#define	DTC_REPEAT_BLOCK_MAX_LENGTH (0x100)
#define	DTC_NORMAL_MAX_LENGTH (0x10000)

Functions

ssp_err_t	R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) Set transfer data in the vector table and enable transfer in ICU. Implements transfer_api_t::open. More...
ssp_err_t	R_DTC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers) Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset. More...
ssp_err_t	R_DTC_Start (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) Start transfer. Implements transfer_api_t::start. More...
ssp_err_t	R_DTC_Stop (transfer_ctrl_t *const p_ctrl) Placeholder for unsupported stop function. Implements transfer_api_t::stop. More...
ssp_err_t	R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl) Enable transfer and ELC events if they are used for software start. Implements transfer_api_t::enable. More...
ssp_err_t	R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl) Disable transfer. Implements transfer_api_t::disable. More...

ssp_err_t	R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info) Set driver specific information. Implements transfer_api_t::infoGet. More...
ssp_err_t	R_DTC_Close (transfer_ctrl_t *const p_api_ctrl) Disables transfer in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close. More...
ssp_err_t	R_DTC_VersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. Implements transfer_api_t::versionGet. More...
ssp_err_t	R_DTC_BlockReset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const length, transfer_size_t size, uint16_t const num_transfers) BlockReset transfer source, destination, length and number of transfers. Implements transfer_api_t::blockReset. More...
ssp_err_t	R_DTC_Stop_ActivationRequest (transfer_ctrl_t *const p_api_ctrl) Placeholder for unsupported DummyTransfer function. Implements transfer_api_t::Stop_ActivationRequest. More...

Detailed Description

Driver for the Data Transfer Controller (DTC).

Summary

Extends Transfer Interface.

The Data Transfer Controller allows data transfers to occur in place of or in addition to any interrupt. It does not support data transfers using software start.

Note

The transfer length is limited to 256 (8 bits) in TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT.

Macro Definition Documentation

◆ DTC_NORMAL_MAX_LENGTH

#define DTC_NORMAL_MAX_LENGTH (0x10000)
Length limited to 65536 transfers for normal mode

◆ DTC_REPEAT_BLOCK_MAX_LENGTH

#define DTC_REPEAT_BLOCK_MAX_LENGTH (0x100)
Length limited to 256 transfers for repeat and block mode

Function Documentation

◆ R_DTC_BlockReset()

ssp_err_t R_DTC_BlockReset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *length*, transfer_size_t *size*, uint16_t const *num_transfers*)

BlockReset transfer source, destination, length and number of transfers. Implements transfer_api_t::blockReset.

Return values

SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	If mode is other than Block Transfer Mode.
SSP_ERR_NOT_ENABLED	Enable failed due to an invalid input parameter: • Transfer source must not be NULL. • Transfer destination must not be NULL.

Disable transfers on this activation source.

Disable read skip prior to modifying settings. It will be enabled later.

Reset transfer based on input parameters.

Enables transfers on this activation source.

Enable read skip after all settings are complete.

◆ R_DTC_Close()

ssp_err_t R_DTC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disables transfer in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the p_ctrl is not enabled in the BSP.

Clear DTC enable bit in ICU.

Clear pointer in vector table.

◆ R_DTC_Disable()

ssp_err_t R_DTC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer. Implements transfer_api_t::disable.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

Disable transfer.

◆ R_DTC_Enable()

ssp_err_t R_DTC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfer and ELC events if they are used for software start. Implements transfer_api_t::enable.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the p_ctrl is not enabled in the BSP.

Enable transfer.

◆ R_DTC_InfoGet()

ssp_err_t R_DTC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info*)

Set driver specific information. Implements transfer_api_t::infoGet.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

If a transfer is active, store it in p_in_progress.

Transfer information for the activation source is taken from DTC vector table.

Mask out the high byte in case of repeat transfer.

Store maximum transfer length.

◆ R_DTC_Open()

ssp_err_t R_DTC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)

Set transfer data in the vector table and enable transfer in ICU. Implements transfer_api_t::open.

Return values

SSP_SUCCESS	Successful open. Transfer is configured and will start when trigger occurs.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_IN_USE	The BSP hardware lock for the DTC is not available, or the index for this IRQ in the DTC vector table is already configured.
SSP_ERR_HW_LOCKED	DTC hardware resource is locked.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
SSP_ERR_NOT_ENABLED	Auto-enable was requested, but enable failed due to an invalid input parameter.

Make sure the activation source is mapped in the ICU.

Make sure the activation source is not already being used by the DTC.

Configure the DTC transfer. See the hardware manual for details.

Update internal variables.

Mark driver as open by initializing it to "DTC" in its ASCII equivalent.

If auto_enable is true, enable transfer and ELC events if software start is used.

Enable read skip after all settings are complete.

◆ R_DTC_Reset()

ssp_err_t R_DTC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset.

Return values

SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_NOT_ENABLED	Transfer length must not be 0 for repeat and block mode, or enable failed due to an invalid input parameter: • Transfer source must not be NULL. • Transfer destination must not be NULL.

Disable transfers on this activation source.

Disable read skip prior to modifying settings. It will be enabled later.

Reset transfer based on input parameters.

Enables transfers on this activation source.

Enable read skip after all settings are complete.

◆ R_DTC_Start()

ssp_err_t R_DTC_Start (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

Start transfer. Implements transfer_api_t::start.

Return values

SSP_SUCCESS	Transfer started successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	One of the following is invalid: • Handle was not configured for software activation. • Mode not set to TRANSFER_START_MODE_SINGLE. • DTC_SOFTWARE_START_ENABLE set to 0 (disabled) in ssp_cfg/driver/r_dtc_cfg.h.

◆ R_DTC_Stop()

ssp_err_t R_DTC_Stop (transfer_ctrl_t *const *p_ctrl*)

Placeholder for unsupported stop function. Implements transfer_api_t::stop.

Return values

SSP_ERR_UNSUPPORTED	DTC software start is not supported.

Mark the input parameter as unused since this function isn't supported.

◆ R_DTC_Stop_ActivationRequest()

ssp_err_t R_DTC_Stop_ActivationRequest (transfer_ctrl_t *const *p_api_ctrl*)

Placeholder for unsupported DummyTransfer function. Implements transfer_api_t::Stop_ActivationRequest.

Return values

SSP_ERR_UNSUPPORTED	R_DTC_Stop_ActivationRequest function is not implemented, Its just a placeholder .

Mark the input parameter as unused since this function isn't supported.

◆ R_DTC_VersionGet()

ssp_err_t R_DTC_VersionGet (ssp_version_t *const *p_version*)
Set driver version based on compile time macros. Implements transfer_api_t::versionGet. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — An input parameter is invalid. Set driver version based on compile time macros

dtc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DTC

#include <r_dtc.h>

Data Fields

uint32_t	id Driver ID.
elc_event_t	trigger Transfer activation event. Matches event returned by transfer_api_t::infoGet.
IRQn_Type	irq Transfer activation IRQ, does not apply to all HAL drivers.
void(*	p_callback)(transfer_callback_args_t *cb_data)
void const *	p_context

Detailed Description

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

Field Documentation

◆ p_callback

void(* dtc_instance_ctrl_t::p_callback) (transfer_callback_args_t *cb_data)
Callback for transfer end interrupt used for ELC software trigger.

◆ p_context

void const* dtc_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

The documentation for this struct was generated from the following file:

- r_dtc.h

dtc_reg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » DTC

#include <r_dtc.h>

Data Fields

struct {	
union {	
uint8_t MRB	
struct {	
uint8_t DM: 2	
	Transfer Destination Address mode.
uint8_t DTS: 1	
	DTC Transfer Mode Select.
uint8_t DISEL: 1	
	DTC Interrupt Select.

uint8_t CHNS: 1	DTC Chain Transfer Select.
uint8_t CHNE: 1	DTC CHain Transfer Enable.
} MRB_b	
}	
union {	
uint8_t MRA	
struct {	
uint8_t SM: 2	Transfer Source Address mode.
uint8_t SZ: 2	DTC Data Transfer Size.
uint8_t MD: 2	DTC Transfer Mode Select.
} MRA_b	
}	
};	
void *volatile	SAR Source address register.
void *volatile	DAR
struct {	
volatile uint16_t CRB	
union {	

```
    uint16_t   CRA

    struct {
      uint8_t   CRAL
                                  Transfer counter A lower register.

      uint8_t   CRAH
                                  Transfer counter B upper register.

    }   CRA_b

  }

};
```

Detailed Description

DTC Registers. Same as transfer_info_t, but uses register names. Provided as service to typecast transfer_info_t.

Field Documentation

◆ @11

struct { ... }
• Mode registers */

◆ @13

struct { ... }
• Transfer count registers */

◆ CRA

uint16_t dtc_reg_t::CRA
Transfer count register A

◆ CRA_b

struct { ... } dtc_reg_t::CRA_b
• bits */

◆ CRB

volatile uint16_t dtc_reg_t::CRB
Transfer count register B

◆ DAR

void* volatile dtc_reg_t::DAR
Destination address register

◆ MRA

uint8_t dtc_reg_t::MRA
Mode Register A

◆ MRA_b

struct { ... } dtc_reg_t::MRA_b
• MRA bits */

◆ MRB

uint8_t dtc_reg_t::MRB
Mode Register B

◆ MRB_b

struct { ... } dtc_reg_t::MRB_b
• MRB bits */

The documentation for this struct was generated from the following file:

- r_dtc.h

5.1.5.17 ELC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Event Link Controller (ELC). More...

Functions

ssp_err_t	R_ELC_Init (elc_cfg_t const *const p_cfg) Initialize all the links in the Event Link Controller. More...
ssp_err_t	R_ELC_SoftwareEventGenerate (elc_software_event_t event_number) Generate a software event in the Event Link Controller. More...
ssp_err_t	R_ELC_LinkSet (elc_peripheral_t peripheral, elc_event_t signal) Create a single event link. More...
ssp_err_t	R_ELC_LinkBreak (elc_peripheral_t peripheral) Break an event link. More...
ssp_err_t	R_ELC_Enable (void) Enable the operation of the Event Link Controller. More...
ssp_err_t	R_ELC_Disable (void) Disable the operation of the Event Link Controller. More...
ssp_err_t	R_ELC_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...

Detailed Description

Driver for the Event Link Controller (ELC).

This module supports the Event Link Controller (ELC). It implements the following interface: events and peripheral definitions

Function Documentation

◆ R_ELC_Disable()

ssp_err_t R_ELC_Disable (void)

Disable the operation of the Event Link Controller.

Implements elc_api_t::disable

Return values

SSP_SUCCESS	ELC disabled.

Disable the Event Link Controller block.

◆ R_ELC_Enable()

ssp_err_t R_ELC_Enable (void)

Enable the operation of the Event Link Controller.

Implements elc_api_t::enable

Return values

SSP_SUCCESS	ELC enabled.

Enable the Event Link Controller block.

◆ R_ELC_Init()

ssp_err_t R_ELC_Init (elc_cfg_t const *const *p_cfg*)

Initialize all the links in the Event Link Controller.

Implements elc_api_t::init

The configuration structure passed in to this function includes links for every event source included in the ELC and sets them all at once. To set an individual link use R_ELC_LinkSet()

Return values

SSP_SUCCESS	Initialization was successful
SSP_ERR_ASSERTION	p_config was NULL

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Power on ELC

Enable the operation of the Event Link Controller

◆ R_ELC_LinkBreak()

ssp_err_t R_ELC_LinkBreak (elc_peripheral_t *peripheral*)

Break an event link.

Implements elc_api_t::linkBreak

Return values

SSP_SUCCESS	Event link broken

Break a link between a signal and a peripheral.

◆ R_ELC_LinkSet()

ssp_err_t R_ELC_LinkSet (elc_peripheral_t *peripheral*, elc_event_t *signal*)

Create a single event link.

Implements elc_api_t::linkSet

Return values

SSP_SUCCESS	Initialization was successful

Make a link between a signal and a peripheral.

◆ R_ELC_SoftwareEventGenerate()

ssp_err_t R_ELC_SoftwareEventGenerate (elc_software_event_t *event_number*)

Generate a software event in the Event Link Controller.

Implements elc_api_t::softwareEventGenerate

Return values

SSP_SUCCESS	Initialization was successful
SSP_ERR_ASSERTION	Invalid event number

Generate a software event in the Event Link Controller.

◆ R_ELC_VersionGet()

ssp_err_t R_ELC_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Implements elc_api_t::versionGet

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

5.1.5.18 High-performance Flash

Renesas Synergy Software Package Reference » HAL Layer

Driver for the High-performance Flash Memory (S7G2 and S5D9). More...

Data Structures

struct	flash_hp_instance_ctrl_t

Functions

ssp_err_t	R_FLASH_HP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) Initializes the flash peripheral. Implements flash_api_t::open. More...
ssp_err_t	R_FLASH_HP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const

	src_address, uint32_t flash_address, uint32_t const num_bytes) Writes to the specified Code or Data Flash memory area. Implements flash_api_t::write. More...
ssp_err_t	R_FLASH_HP_Read (flash_ctrl_t *const p_api_ctrl, uint8_t *p_dest_address, uint32_t const flash_address, uint32_t const num_bytes) Reads the requested number of bytes from the supplied Data or Code Flash memory address. Implements flash_api_t::read. More...
ssp_err_t	R_FLASH_HP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks) Erases the specified Code or Data Flash blocks. Implements flash_api_t::erase by the block_erase_address. More...
ssp_err_t	R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *p_blank_check_result) Performs a blank check on the specified address area. Implements flash_api_t::blankCheck. More...
ssp_err_t	R_FLASH_HP_StatusGet (flash_ctrl_t *const p_api_ctrl) Query the FLASH peripheral for its status. Implements flash_api_t::statusGet. More...
ssp_err_t	R_FLASH_HP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) Implements flash_api_t::idCodeSet. More...
ssp_err_t	R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) Configure an access window for the Code Flash memory using the provided start and end address. An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block –> last block inclusive. Anything outside this range of Code Flash is then write protected. More...
ssp_err_t	R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)

	Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements flash_api_t::accessWindowClear. More...
ssp_err_t	R_FLASH_HP_Reset (flash_ctrl_t *const p_api_ctrl) Resets the FLASH peripheral. Implements flash_api_t::reset. No attempt is made to grab the Flash software lock before executing the reset since the assumption is that a reset will terminate any existing operation. More...
ssp_err_t	R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window. Implements flash_api_t::startupAreaSelect. More...
ssp_err_t	R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) Indicate to the already open Flash API, that the FCLK has changed since the Open(). This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro. This function uses R_CGC_SystemClockFreqGet() to get the current FCLK frequency. Implements flash_api_t::updateFlashClockFreq. More...
ssp_err_t	R_FLASH_HP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) Returns the information about the flash regions. Implements flash_api_t::infoGet. More...
ssp_err_t	R_FLASH_HP_Close (flash_ctrl_t *const p_api_ctrl) Releases any resources that were allocated by the Open() or any subsequent Flash operations. Implements flash_api_t::close. More...
ssp_err_t	R_FLASH_HP_VersionGet (ssp_version_t *const p_version) This function gets FLASH HAL driver version. More...

Detailed Description

Driver for the High-performance Flash Memory (S7G2 and S5D9).

This module supports the Flash interface for the High Performance FLASH peripheral.

Function Documentation

◆ R_FLASH_HP_AccessWindowClear()

ssp_err_t R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements flash_api_t::accessWindowClear.

Return values

SSP_SUCCESS	Access window successfully removed.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_INVALID_HW_CONDITION	The configuration area cannot be written while DTC/DMAC, EDMAC, LCDC/2DG/JPEG are enabled. This is to prevent prohibited memory accesses while the flash sequencer is running. Refer to the technical update number TN-SY*-A033A/E. This check can be disabled by defining R_FLASH_HP_CHECK_MODULE_STOP_BITS to 0. To disable DMAC/DTC, close all modules using DMAC/DTC and call the following: ssp_err_t err; ssp_feature_t ssp_feature = {{(ssp_ip_t) 0U}}; ssp_feature.id = SSP_IP_DTC; err = R_BSP_ModuleStopAlways(&ssp_feature);
SSP_ERR_INVALID_LINKED_ADDRESS	HW_FLASH_HP_configurationSet is linked to an invalid region of memory. This function must not reside in prohibited regions of memory. It is recommended that the following changes be made to the linker script. For information about prohibited regions refer to the S5 series flash technical update. Technical Update number TN-SY*-A033A/E Recommended Changes: For GCC: Add the following immediately after "__data_start__ = .": . = ALIGN(4); __Code_In_RAM_Start = .; KEEP(*(.code_in_ram*)) __Code_In_RAM_End = .; For IAR: Change: place at start of RAM_region { block START_OF_RAM }; To: define block START_OF_RAM with fixed order { rw section .ssp_dtc_vector_table, block RAM_CODE }; place at start of RAM_region { block START_OF_RAM };

< For consistency with _LP API we return error if Code Flash not enabled

◆ R_FLASH_HP_AccessWindowSet()

ssp_err_t R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr*)

Configure an access window for the Code Flash memory using the provided start and end address. An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block –> last block inclusive. Anything outside this range of Code Flash is then write protected.

Note

If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_HP_AccessWindowClear().

Implements flash_api_t::accessWindowSet.

Return values

SSP_SUCCESS	Access window successfully configured.
SSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_INVALID_HW_CONDITION	The configuration area cannot be written while DTC/DMAC, EDMAC, LCDC/2DG/JPEG are enabled. This is to prevent prohibited memory accesses while the flash sequencer is running. Refer to the technical update number TN-SY*-A033A/E. This check can be disabled by defining R_FLASH_HP_CHECK_MODULE_STOP_BITS to 0. To disable DMAC/DTC, close all modules using DMAC/DTC and call the following: ssp_err_t err; ssp_feature_t ssp_feature = {{(ssp_ip_t) 0U}}; ssp_feature.id = SSP_IP_DTC; err = R_BSP_ModuleStopAlways(&ssp_feature);
SSP_ERR_INVALID_LINKED_ADDRESS	HW_FLASH_HP_configurationSet is linked to an invalid region of memory. This function must not reside in prohibited regions of memory. It is recommended that the following changes be made to the linker script. For information about prohibited regions refer to the S5 series flash technical update. Technical Update number TN-SY*-A033A/E Recommended Changes: For

	GCC: Add the following immediately after "__data_start__ = .": . = ALIGN(4); __Code_In_RAM_Start = .; KEEP(*(.code_in_ram*)) __Code_In_RAM_End = .; For IAR: Change: place at start of RAM_region { block START_OF_RAM }; To: define block START_OF_RAM with fixed order { rw section .ssp_dtc_vector_table, block RAM_CODE }; place at start of RAM_region { block START_OF_RAM };

< For consistency with _LP API we return error if Code Flash not enabled

Return status.

◆ R_FLASH_HP_BlankCheck()

ssp_err_t R_FLASH_HP_BlankCheck (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result*)

Performs a blank check on the specified address area. Implements flash_api_t::blankCheck.

Return values

SSP_SUCCESS	Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode).
SSP_ERR_INVALID_ADDRESS	Invalid data flash address was input.
SSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
SSP_ERR_IN_USE	Other flash operation in progress or API not initialized.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

Get the block information for this address. If failure return error.

Setup blank check. If failure return error.

Initiate the Blank Check operation

Complete the flash operation.

◆ R_FLASH_HP_Close()

ssp_err_t R_FLASH_HP_Close (flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent Flash operations.
Implements flash_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

Return the hardware lock for the Flash

Disable interrupts

Release the lock

Close the API

◆ R_FLASH_HP_Erase()

ssp_err_t R_FLASH_HP_Erase (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks*)

Erases the specified Code or Data Flash blocks. Implements flash_api_t::erase by the block_erase_address.

Return values

SSP_SUCCESS	Successful open.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
SSP_ERR_INVALID_ADDRESS	Invalid address specified
SSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to erase CF was requested.

Get the block information for this address. If failure return error.

Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE. If failure return error.

Configure current operation parameters based on user input.

Erase the Blocks. If not a DF BGO erase then exit PE mode and return status.

Complete the flash operation.

◆ R_FLASH_HP_IdCodeSet()

ssp_err_t R_FLASH_HP_IdCodeSet (flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode*)

Implements flash_api_t::idCodeSet.

Return values

SSP_SUCCESS	ID Code successfully configured.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_INVALID_HW_CONDITION	The configuration area cannot be written while DTC/DMAC, EDMAC, LCDC/2DG/JPEG are enabled. This is to prevent prohibited memory accesses while the flash sequencer is running. Refer to the technical update number TN-SY*-A033A/E. This check can be disabled by defining R_FLASH_HP_CHECK_MODULE_STOP_BITS to 0. To disable DMAC/DTC, close all modules using DMAC/DTC and call the following: ssp_err_t err; ssp_feature_t ssp_feature = {{(ssp_ip_t) 0U}}; ssp_feature.id = SSP_IP_DTC; err = R_BSP_ModuleStopAlways(&ssp_feature);
SSP_ERR_INVALID_LINKED_ADDRESS	HW_FLASH_HP_configurationSet is linked to an invalid region of memory. This function must not reside in prohibited regions of memory. It is recommended that the following changes be made to the linker script. For information about prohibited regions refer to the S5 series flash technical update. Technical Update number TN-SY*-A033A/E Recommended Changes: For GCC: Add the following immediately after "__data_start__ = .": . = ALIGN(4); __Code_In_RAM_Start = .; KEEP(*(.code_in_ram*)) __Code_In_RAM_End = .; For IAR: Change: place at start of RAM_region { block START_OF_RAM }; To: define block START_OF_RAM with fixed order { rw section .ssp_dtc_vector_table, block RAM_CODE }; place at start of RAM_region { block START_OF_RAM };

Return status.

◆ R_FLASH_HP_InfoGet()

ssp_err_t R_FLASH_HP_InfoGet (flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info*)

Returns the information about the flash regions. Implements flash_api_t::infoGet.

Return values

SSP_SUCCESS	Successful retrieved the request information.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

Eliminate warning if parameter checking is disabled.

Copy information about the code and data flash to the user structure.

◆ R_FLASH_HP_Open()

ssp_err_t R_FLASH_HP_Open (flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg*)

Initializes the flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash. It first copies the FCU firmware to FCURAM and sets the FCU Clock based on the current FCLK frequency. In addition, if Code Flash programming is enabled, the API code responsible for Code Flash programming will be copied to RAM.

This function must be called once prior to calling any other FLASH API functions. If a user supplied callback function is supplied, then the Flash Ready and Error interrupts will be configured to call the users callback routine with an Event type describing the source of the interrupt.

Note

Providing a callback function in the supplied p_cfg->callback field, automatically configures the Flash for Data Flash to operate in non-blocking (BGO) mode.

Subsequent to a successful Open(), the Flash is ready to process additional Flash commands.

Return values

SSP_SUCCESS	Initialization was successful and timer has started.
SSP_FLASH_ERR_FAILURE	Failed to successfully enter Programming/Erase mode.
SSP_ERR_TIMEOUT	Timed out waiting for FCU to be ready.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg or problem getting FMI info.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
SSP_ERR_HW_LOCKED	FLASH peripheral has already been initialized and is in use.

Setup the flash FMI and acquire the hardware lock. If failure return error.

Allow Initialization if not initialized or if no operation is ongoing and re-initialization is desired

Acquire the flash hp software lock.

Set the parameters struct based on the user supplied settings

If BGO is enabled for data flash configure the callback and enable flash iterrupts. Otherwise disable interrupts.

Flash open setup.

Save callback function pointer

◆ R_FLASH_HP_Read()

ssp_err_t R_FLASH_HP_Read (flash_ctrl_t *const *p_api_ctrl*, uint8_t * *p_dest_address*, uint32_t const *flash_address*, uint32_t const *num_bytes*)

Reads the requested number of bytes from the supplied Data or Code Flash memory address. Implements flash_api_t::read.

Note

This function is provided simply for the purposes of maintaining a complete interface. It is possible (and recommended), to read Flash memory directly.

Return values

SSP_SUCCESS	Operation successful.
SSP_ERR_INVALID_ADDRESS	Invalid Flash address was supplied.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_dest_address

Eliminate warning if parameter checking is disabled.

Copy data to the destination buffer.

◆ R_FLASH_HP_Reset()

ssp_err_t R_FLASH_HP_Reset (flash_ctrl_t *const *p_api_ctrl*)

Resets the FLASH peripheral. Implements flash_api_t::reset. No attempt is made to grab the Flash software lock before executing the reset since the assumption is that a reset will terminate any existing operation.

Return values

SSP_SUCCESS	Flash circuit successfully reset.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

Reset the flash.

Release the flash.

◆ R_FLASH_HP_StartUpAreaSelect()

ssp_err_t R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary*)

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window. Implements flash_api_t::startupAreaSelect.

Return values

SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_HW_CONDITION	The configuration area cannot be written while DTC/DMAC, EDMAC, LCDC/2DG/JPEG are enabled. This is to prevent prohibited memory accesses while the flash sequencer is running. Refer to the technical update number TN-SY*-A033A/E. This check can be disabled by defining R_FLASH_HP_CHECK_MODULE_STOP_BITS to 0. To disable DMAC/DTC, close all modules using DMAC/DTC and call the following: ssp_err_t err; ssp_feature_t ssp_feature = {{(ssp_ip_t) 0U}}; ssp_feature.id = SSP_IP_DTC; err = R_BSP_ModuleStopAlways(&ssp_feature);
SSP_ERR_INVALID_LINKED_ADDRESS	HW_FLASH_HP_configurationSet is linked to an invalid region of memory. This function must not reside in prohibited regions of memory. It is recommended that the following changes be made to the linker script. For information about prohibited regions refer to the S5 series flash technical update. Technical Update number TN-SY*-A033A/E Recommended Changes: For GCC: Add the following immediately after "__data_start__ = .": . = ALIGN(4); __Code_In_RAM_Start = .; KEEP(*(.code_in_ram*)) __Code_In_RAM_End = .; For IAR: Change: place at start of RAM_region { block START_OF_RAM }; To: define block START_OF_RAM with fixed order { rw section .ssp_dtc_vector_table, block RAM_CODE }; place at start of RAM_region { block START_OF_RAM };

Enter PE Mode.

Swap the block temporarily or permantly based on caller.

If failure reset the flash.

Release the flash and return status.

◆ R_FLASH_HP_StatusGet()

ssp_err_t R_FLASH_HP_StatusGet (flash_ctrl_t *const *p_api_ctrl*)

Query the FLASH peripheral for its status. Implements flash_api_t::statusGet.

Return values

SSP_SUCCESS	FLASH peripheral is ready to use.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

Eliminate warning if parameter checking is disabled.

Return flash status

◆ R_FLASH_HP_UpdateFlashClockFreq()

ssp_err_t R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API, that the FCLK has changed since the Open(). This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro. This function uses R_CGC_SystemClockFreqGet() to get the current FCLK frequency. Implements flash_api_t::updateFlashClockFreq.

Return values

SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

Lock the flash state. If failure return error.

Flash Setup

Release the flash.

◆ R_FLASH_HP_VersionGet()

ssp_err_t R_FLASH_HP_VersionGet (ssp_version_t *const *p_version*)

This function gets FLASH HAL driver version.

Return values

SSP_SUCCESS	- operation performed successfully

Note

This function is reentrant.

Copy the version information.

◆ R_FLASH_HP_Write()

ssp_err_t R_FLASH_HP_Write (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes*)

Writes to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Return values

SSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
SSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window.
SSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
SSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
SSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
SSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to write CF was requested.

Get the block information for this address. If failure return error.

Initiate the write operation, may return SSP_ERR_IN_USE via setup_for_pe_mode()

Execute a reset if any error, release the state if not BGO

flash_hp_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » High-performance Flash

#include <r_flash_hp.h>

Data Fields

uint32_t	opened To check whether api has been opened or not.
R_FACI_Type *	p_reg Base address of flash registers.
bsp_cache_state_t	cache_state User Callback function. More...
IRQn_Type	irq Flash ready interrupt number.
IRQn_Type	err_irq Flash error interrupt number.

Detailed Description

Flash HP instance control block. DO NOT INITIALIZE.

Field Documentation

◆ cache_state

bsp_cache_state_t flash_hp_instance_ctrl_t::cache_state
User Callback function. Used to disable and then restore Flash Cache while API is open.

The documentation for this struct was generated from the following file:

- r_flash_hp.h

5.1.5.19 Low Power Flash

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Low power Flash Memory (S3A7 and S124). More...

Data Structures

struct	flash_lp_instance_ctrl_t

Functions

ssp_err_t	R_FLASH_LP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) Initialize the Low Power flash peripheral. Implements flash_api_t::open. More...
ssp_err_t	R_FLASH_LP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes) Write to the specified Code or Data Flash memory area. Implements flash_api_t::write. More...
ssp_err_t	R_FLASH_LP_Read (flash_ctrl_t *const p_api_ctrl, uint8_t *p_dest_address, uint32_t const flash_address, uint32_t const num_bytes) Read the requested number of bytes from the supplied Data or Code Flash address. Implements flash_api_t::read. More...
ssp_err_t	R_FLASH_LP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks) Erase the specified Code or Data Flash blocks. Implements flash_api_t::erase. More...
ssp_err_t	R_FLASH_LP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *p_blank_check_result) Perform a blank check on the specified address area. Implements flash_api_t::blankCheck. More...
ssp_err_t	R_FLASH_LP_StatusGet (flash_ctrl_t *const p_api_ctrl) Query the FLASH for its status. Implements flash_api_t::statusGet. More...
ssp_err_t	R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) Configure an access window for the Code Flash memory. Implements flash_api_t::accessWindowSet. More...

ssp_err_t	R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) Remove any access window that is configured in the Code Flash. Implements flash_api_t::accessWindowClear. On successful return from this call all Code Flash is writable. More...
ssp_err_t	R_FLASH_LP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) Write the ID code provided to the id code registers. Implements flash_api_t::idCodeSet. More...
ssp_err_t	R_FLASH_LP_Reset (flash_ctrl_t *const p_api_ctrl) Reset the FLASH peripheral. Implements flash_api_t::reset. More...
ssp_err_t	R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) Select which block is used as the startup area block. Implements flash_api_t::startupAreaSelect. More...
ssp_err_t	R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) Indicate to the already open Flash API that the FCLK has changed. Implements r_flash_t::updateFlashClockFreq. More...
ssp_err_t	R_FLASH_LP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) Returns the information about the flash regions. Implements flash_api_t::infoGet. More...
ssp_err_t	R_FLASH_LP_Close (flash_ctrl_t *const p_api_ctrl) Release any resources that were allocated by the Flash API. Implements flash_api_t::close. More...
ssp_err_t	R_FLASH_LP_VersionGet (ssp_version_t *const p_version) Get FLASH HAL driver version. More...

Detailed Description

Driver for the Low power Flash Memory (S3A7 and S124).

This module supports the Flash interface for the Low-power FLASH peripheral.

Function Documentation

◆ R_FLASH_LP_AccessWindowClear()

ssp_err_t R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is configured in the Code Flash. Implements flash_api_t::accessWindowClear. On successful return from this call all Code Flash is writable.

Return values

SSP_SUCCESS	Access window successfully removed.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

< Return error if Code Flash not enabled

◆ R_FLASH_LP_AccessWindowSet()

ssp_err_t R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr*)

Configure an access window for the Code Flash memory. Implements flash_api_t::accessWindowSet .

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block –> last block inclusive. Anything outside this range of Code Flash is then write protected. As an example, if you wanted to place an accesswindow on Code Flash Blocks 0 and 1, such that only those two blocks were writable, you would need to specify (address in block 0, address in block 2) as the respective start and end address.

Note

If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_LP_AccessWindowClear().

The invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Return values

SSP_SUCCESS	Access window successfully configured.
SSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

If not code flash return error.

◆ R_FLASH_LP_BlankCheck()

ssp_err_t R_FLASH_LP_BlankCheck (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result*)

Perform a blank check on the specified address area. Implements flash_api_t::blankCheck.

The minimum/maximum number of bytes, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information. The number of bytes for Data Flash blank checking must be between (1 and FLASH_DATA_BLANK_CHECK_MAX). The number of bytes for Code Flash blank checking must be between (1 and FLASH_CODE_BLANK_CHECK_MAX).

Return values

SSP_SUCCESS	Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode).
SSP_ERR_INVALID_ADDRESS	Invalid data flash address was input
SSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to Blank Check CF was requested.

Get the block information for this address. If failure return error.

Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE

Initiate the Blank Check operation

If failure reset the flash.

If the current operation is not a BGO blank check release the flash.

◆ R_FLASH_LP_Close()

ssp_err_t R_FLASH_LP_Close (flash_ctrl_t *const *p_api_ctrl*)

Release any resources that were allocated by the Flash API. Implements flash_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

Unlock the flash hardware.

Disable the flash interrupt.

Mark the control block as closed.

Release the lock

◆ R_FLASH_LP_Erase()

ssp_err_t R_FLASH_LP_Erase (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks*)

Erase the specified Code or Data Flash blocks. Implements flash_api_t::erase.

The minimum/maximum number of blocks, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Return values

SSP_SUCCESS	Successful open.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
SSP_ERR_INVALID_ADDRESS	Invalid address specified
SSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to erase CF was requested.

Get the block information for this address. If failure return error.

Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE

If successful

Configure the current parameters based on if the operation is for code flash or data flash.

If this is a request to erase Data Flash configure BGO mode if it is enabled.

Initiate the flash erase.

If in non-BGO mode, the current operation is complete. Exit PE mode and return status

Complete the flash operation.

◆ R_FLASH_LP_IdCodeSet()

ssp_err_t R_FLASH_LP_IdCodeSet (flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode*)

Write the ID code provided to the id code registers. Implements flash_api_t::idCodeSet.

Return values

SSP_SUCCESS	ID code successfully configured.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
SSP_ERR_TIMEOUT	Timed out waiting for completion of extra command.

Return status.

◆ R_FLASH_LP_InfoGet()

ssp_err_t R_FLASH_LP_InfoGet (flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info*)

Returns the information about the flash regions. Implements flash_api_t::infoGet.

Return values

SSP_SUCCESS	Successful retrieved the request information.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

Copy the region data to the info structure.

◆ R_FLASH_LP_Open()

ssp_err_t R_FLASH_LP_Open (flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg*)

Initialize the Low Power flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash. It first copies the FCU firmware to FCURAM and sets the FCU Clock based on the current FCLK frequency. In addition, if Code Flash programming is enabled, the API code responsible for Code Flash programming will be copied to RAM.

This function must be called once prior to calling any other FLASH API functions. If a user supplied

callback function is supplied, then the Flash Ready interrupt will be configured to call the users callback routine with an Event type describing the source of the interrupt for Data Flash operations. Subsequent to successfully completing this call p_ctrl->opened will be true.

Note

Providing a callback function in the supplied p_cfg->callback field, automatically configures the Flash for Data Flash to operate in non-blocking (BGO) mode.

Subsequent to a successful Open(), the Flash is ready to process additional Flash commands.

Return values

SSP_SUCCESS	Initialization was successful and timer has started.
SSP_FLASH_ERR_FAILURE	Failed to successfully enter Programming/Erase mode.
SSP_ERR_TIMEOUT	Timed out waiting for FCU to be ready.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg or problem getting FMI info.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
SSP_ERR_FCLK	FCLK must be a minimum of 1 MHz for Flash operations.
SSP_ERR_IN_USE	Flash Open() has already been called.
SSP_ERR_HW_LOCKED	Flash module unable to get the Hardware lock for the Flash LP Periperhal.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

If null pointers return error.

If open return error.

Setup flash FMI. If failure return error.

Allow Initialization if not initialized or if no operation is ongoing and re-initialization is desired

Acquire the software lock. If failure return error.

Set the parameters struct based on the user supplied settings

Setup the Flash interrupt callback based on the caller's info. If the Flash interrupt is not enabled in the BSP then this will return SSP_ERR_IRQ_BSP_DISABLED

Make sure Flash interrupts are disabled, they are only used in BGO mode

Setup the flash.

Save callback function pointer

◆ R_FLASH_LP_Read()

ssp_err_t R_FLASH_LP_Read (flash_ctrl_t *const *p_api_ctrl*, uint8_t * *p_dest_address*, uint32_t const *flash_address*, uint32_t const *num_bytes*)

Read the requested number of bytes from the supplied Data or Code Flash address. Implements flash_api_t::read.

The minimum/maximum number of blocks, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Note

This function is provided simply for the purposes of maintaining a complete interface. It is possible (and recommended), to read Flash memory directly.

Return values

SSP_SUCCESS	Operation successful.
SSP_ERR_INVALID_ADDRESS	Invalid Flash address was supplied.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_dest_address
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

Copy the data to the destination buffer.

◆ R_FLASH_LP_Reset()

ssp_err_t R_FLASH_LP_Reset (flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to grab the Flash software lock before executing the reset since the assumption is that a reset will terminate any existing operation.

Return values

SSP_SUCCESS	Flash circuit successfully reset.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

Reset the flash.

Release the flash.

◆ R_FLASH_LP_StartUpAreaSelect()

ssp_err_t R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary*)

Select which block is used as the startup area block. Implements flash_api_t::startupAreaSelect.

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window.

Return values

SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

If the swap type is BTFLG and the operation is temporary there's nothing to do. Return success.

Update Flash state and enter the respective Code or Data Flash P/E mode.

If successful call the temporary or boot startup area set functions depending on the users flag.

If successful return to read mode otherwise reset the flash.

Release the flash.

◆ R_FLASH_LP_StatusGet()

ssp_err_t R_FLASH_LP_StatusGet (flash_ctrl_t *const *p_api_ctrl*)

Query the FLASH for its status. Implements flash_api_t::statusGet.

Return values

SSP_SUCCESS	Flash is ready and available to accept commands.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

Return flash status

◆ R_FLASH_LP_UpdateFlashClockFreq()

ssp_err_t R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements r_flash_t::updateFlashClockFreq.

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro. This function uses R_CGC_SystemClockFreqGet() to get the current FCLK frequency.

Return values

SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

Lock the flash state. If failure return error.

Setup the Flash.

Release the flash.

◆ R_FLASH_LP_VersionGet()

ssp_err_t R_FLASH_LP_VersionGet (ssp_version_t *const *p_version*)

Get FLASH HAL driver version.

Return values

SSP_SUCCESS	- operation performed successfully

Note

This function is reentrant.

Return the version id of the flash lp module.

◆ R_FLASH_LP_Write()

ssp_err_t R_FLASH_LP_Write (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes*)

Write to the specified Code or Data Flash memory area. Implements flash_api_t::write.

The minimum/maximum number of bytes, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Return values

SSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
SSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window.
SSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
SSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
SSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
SSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to write CF was requested.

Get the block information for this address. If failure return error.

Initiate the write operation.

Return status.

flash_lp_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Low Power Flash

#include <r_flash_lp.h>

Data Fields

uint32_t	opened To check whether api has been opened or not.
void *	p_reg Base address of flash registers.
bsp_cache_state_t	cache_state Used to disable and then restore Flash Cache while API is open.
IRQn_Type	irq Flash ready interrupt number.

Detailed Description

Flash instance control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_flash_lp.h

5.1.5.20 FMI

Renesas Synergy Software Package Reference » HAL Layer

Driver for accessing Factory MCU Information (FMI). More...

Detailed Description

Driver for accessing Factory MCU Information (FMI).

Read Factory MCU Information flash memory.

5.1.5.21 GLCDC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Graphics LCD Controller (GLCDC). More...

Data Structures

struct	glcd_instance_ctrl_t
struct	glcd_cfg_t
struct	glcd_ctrl_t

Enumerations

enum	glcd_clk_src_t { GLCD_CLK_SRC_INTERNAL, GLCD_CLK_SRC_EXTERNAL }
enum	glcd_panel_clk_div_t { GLCD_PANEL_CLK_DIVISOR_1 = 1, GLCD_PANEL_CLK_DIVISOR_2 = 2, GLCD_PANEL_CLK_DIVISOR_3 = 3, GLCD_PANEL_CLK_DIVISOR_4 = 4, GLCD_PANEL_CLK_DIVISOR_5 = 5, GLCD_PANEL_CLK_DIVISOR_6 = 6, GLCD_PANEL_CLK_DIVISOR_7 = 7, GLCD_PANEL_CLK_DIVISOR_8 = 8, GLCD_PANEL_CLK_DIVISOR_9 = 9, GLCD_PANEL_CLK_DIVISOR_12 = 12, GLCD_PANEL_CLK_DIVISOR_16 = 16, GLCD_PANEL_CLK_DIVISOR_24 = 24, GLCD_PANEL_CLK_DIVISOR_32 = 32 }
enum	glcd_tcon_pin_t { GLCD_TCON_PIN_NONE = -1, GLCD_TCON_PIN_0, GLCD_TCON_PIN_1, GLCD_TCON_PIN_2, GLCD_TCON_PIN_3 }
enum	glcd_bus_arbitration_t { GLCD_BUS_ARBITRATION_ROUNDROBIN, GLCD_BUS_ARBITRATION_FIX_PRIORITY }
enum	glcd_correction_proc_order_t { GLCD_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA, GLCD_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST }
enum	glcd_tcon_signal_select_t { GLCD_TCON_SIGNAL_SELECT_STVA_VS = 0, GLCD_TCON_SIGNAL_SELECT_STVB_VE = 1,

	GLCD_TCON_SIGNAL_SELECT_STHA_HS = 2, GLCD_TCON_SIGNAL_SELECT_STHB_HE = 3, GLCD_TCON_SIGNAL_SELECT_DE = 7 }
enum	glcd_clut_plane_t { GLCD_CLUT_PLANE_0 = 0, GLCD_CLUT_PLANE_1 = 1 }
enum	glcd_dithering_mode_t { GLCD_DITHERING_MODE_TRUNCATE = 0, GLCD_DITHERING_MODE_ROUND_OFF = 1, GLCD_DITHERING_MODE_2X2PATTERN = 2, GLCD_DITHERING_MODE_SETTING_MAX }
enum	glcd_dithering_pattern_t { GLCD_DITHERING_PATTERN_00 = 0, GLCD_DITHERING_PATTERN_01 = 1, GLCD_DITHERING_PATTERN_10 = 2, GLCD_DITHERING_PATTERN_11 = 3 }

enum	glcd_input_interface_format_t {
	GLCD_INPUT_INTERFACE_FORMAT_RGB565 = 0,
	GLCD_INPUT_INTERFACE_FORMAT_RGB888 = 1,
	GLCD_INPUT_INTERFACE_FORMAT_ARGB1555 = 2,
	GLCD_INPUT_INTERFACE_FORMAT_ARGB4444 = 3,
	GLCD_INPUT_INTERFACE_FORMAT_ARGB8888 = 4,
	GLCD_INPUT_INTERFACE_FORMAT_CLUT8 = 5,
	GLCD_INPUT_INTERFACE_FORMAT_CLUT4 = 6,
	GLCD_INPUT_INTERFACE_FORMAT_CLUT1 = 7
	}

enum	glcd_output_interface_format_t { GLCD_OUTPUT_INTERFACE_FORMAT_RGB888 = 0, GLCD_OUTPUT_INTERFACE_FORMAT_RGB666 = 1, GLCD_OUTPUT_INTERFACE_FORMAT_RGB565 = 2, GLCD_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB = 3 }
enum	glcd_dithering_output_format_t { GLCD_DITHERING_OUTPUT_FORMAT_RGB888 = 0, GLCD_DITHERING_OUTPUT_FORMAT_RGB666 = 1, GLCD_DITHERING_OUTPUT_FORMAT_RGB565 = 2 }

Functions

ssp_err_t	R_GLCD_Open (display_ctrl_t *const p_api_ctrl, display_cfg_t const *const p_cfg) Open GLCDC module. More...
ssp_err_t	R_GLCD_Close (display_ctrl_t *const p_api_ctrl) Close GLCDC module. More...
ssp_err_t	R_GLCD_Start (display_ctrl_t *const p_api_ctrl)

	Start GLCDC module. More...
ssp_err_t	R_GLCD_Stop (display_ctrl_t *const p_api_ctrl) Stop GLCDC module. More...
ssp_err_t	R_GLCD_LayerChange (display_ctrl_t const *const p_api_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame) Change layer parameters of GLCDC module at runtime. More...
ssp_err_t	R_GLCD_ColorCorrection (display_ctrl_t const *const p_api_ctrl, display_correction_t const *const p_correction) Perform color correction by GLCDC module. More...
ssp_err_t	R_GLCD_ClutUpdate (display_ctrl_t const *const p_api_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame) Update Color Look Up Table of GLCDC module. More...
ssp_err_t	R_GLCD_StatusGet (display_ctrl_t const *const p_api_ctrl, display_status_t *const p_status) Get status of GLCDC module. More...
ssp_err_t	R_GLCD_VersionGet (ssp_version_t *p_version) Get version of R_GLCDC module. More...

Detailed Description

Driver for the Graphics LCD Controller (GLCDC).

Summary

Implements Display Interface. This module supports the Graphics LCD Controller (GLCDC). It implements the display interface and drives LCD panels connected to the GLCDC pins.

Enumeration Type Documentation

◆ glcd_bus_arbitration_t

enum glcd_bus_arbitration_t	
Bus Arbitration setting	
Enumerator	
GLCD_BUS_ARBITRATION_ROUNDROBIN	Round robin.
GLCD_BUS_ARBITRATION_FIX_PRIORITY	Fixed.

◆ glcd_clk_src_t

enum glcd_clk_src_t	
Clock source select	
Enumerator	
GLCD_CLK_SRC_INTERNAL	Internal.
GLCD_CLK_SRC_EXTERNAL	External.

◆ glcd_clut_plane_t

enum glcd_clut_plane_t	
Clock phase adjustment for serial RGB output	
Enumerator	
GLCD_CLUT_PLANE_0	GLCD CLUT plane 0.
GLCD_CLUT_PLANE_1	GLCD CLUT plane 1.

◆ glcd_correction_proc_order_t

enum glcd_correction_proc_order_t	
Correction circuit sequence control	
Enumerator	
GLCD_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA	Brightness -> contrast -> gamma correction.
GLCD_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST	Gamma correction -> brightness -> contrast.

◆ glcd_dithering_mode_t

enum glcd_dithering_mode_t	
Dithering mode	
Enumerator	
GLCD_DITHERING_MODE_TRUNCATE	No dithering (truncate)
GLCD_DITHERING_MODE_ROUND_OFF	Dithering with round off.
GLCD_DITHERING_MODE_2X2PATTERN	Dithering with 2x2 pattern.
GLCD_DITHERING_MODE_SETTING_MAX	Setting prohibited.

◆ glcd_dithering_output_format_t

enum glcd_dithering_output_format_t	
Dithering output format	
Enumerator	
GLCD_DITHERING_OUTPUT_FORMAT_RGB888	Dithering output format RGB888.
GLCD_DITHERING_OUTPUT_FORMAT_RGB666	Dithering output format RGB666.
GLCD_DITHERING_OUTPUT_FORMAT_RGB565	Dithering output format RGB565.

◆ glcd_dithering_pattern_t

enum glcd_dithering_pattern_t	
Dithering mode	
Enumerator	
GLCD_DITHERING_PATTERN_00	2x2 pattern '00'
GLCD_DITHERING_PATTERN_01	2x2 pattern '01'
GLCD_DITHERING_PATTERN_10	2x2 pattern '10'
GLCD_DITHERING_PATTERN_11	2x2 pattern '11'

◆ glcd_input_interface_format_t

enum glcd_input_interface_format_t	
Output interface format	
Enumerator	
GLCD_INPUT_INTERFACE_FORMAT_RGB565	Input interface format RGB565.
GLCD_INPUT_INTERFACE_FORMAT_RGB888	Input interface format RGB888.
GLCD_INPUT_INTERFACE_FORMAT_ARGB1555	Input interface format ARGB1555.
GLCD_INPUT_INTERFACE_FORMAT_ARGB4444	Input interface format ARGB4444.
GLCD_INPUT_INTERFACE_FORMAT_ARGB8888	Input interface format ARGB8888.
GLCD_INPUT_INTERFACE_FORMAT_CLUT8	Input interface format CLUT8.
GLCD_INPUT_INTERFACE_FORMAT_CLUT4	Input interface format CLUT4.
GLCD_INPUT_INTERFACE_FORMAT_CLUT1	Input interface format CLUT1.

◆ glcd_output_interface_format_t

enum glcd_output_interface_format_t	
Output interface format	
Enumerator	
GLCD_OUTPUT_INTERFACE_FORMAT_RGB888	Output interface format RGB888.
GLCD_OUTPUT_INTERFACE_FORMAT_RGB666	Output interface format RGB666.
GLCD_OUTPUT_INTERFACE_FORMAT_RGB565	Output interface format RGB565.
GLCD_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB	Output interface format Serial RGB.

◆ glcd_panel_clk_div_t

enum glcd_panel_clk_div_t	
Clock frequency division ratio	
Enumerator	
GLCD_PANEL_CLK_DIVISOR_1	Division Ratio 1/1.
GLCD_PANEL_CLK_DIVISOR_2	Division Ratio 1/2.
GLCD_PANEL_CLK_DIVISOR_3	Division Ratio 1/3.
GLCD_PANEL_CLK_DIVISOR_4	Division Ratio 1/4.
GLCD_PANEL_CLK_DIVISOR_5	Division Ratio 1/5.
GLCD_PANEL_CLK_DIVISOR_6	Division Ratio 1/6.
GLCD_PANEL_CLK_DIVISOR_7	Division Ratio 1/7.
GLCD_PANEL_CLK_DIVISOR_8	Division Ratio 1/8.
GLCD_PANEL_CLK_DIVISOR_9	Division Ratio 1/9.
GLCD_PANEL_CLK_DIVISOR_12	Division Ratio 1/12.
GLCD_PANEL_CLK_DIVISOR_16	Division Ratio 1/16.
GLCD_PANEL_CLK_DIVISOR_24	Division Ratio 1/24.
GLCD_PANEL_CLK_DIVISOR_32	Division Ratio 1/32.

◆ glcd_tcon_pin_t

enum glcd_tcon_pin_t	
LCD TCON output pin select	
Enumerator	
GLCD_TCON_PIN_NONE	No output.
GLCD_TCON_PIN_0	LCD_TCON0.
GLCD_TCON_PIN_1	LCD_TCON1.
GLCD_TCON_PIN_2	LCD_TCON2.
GLCD_TCON_PIN_3	LCD_TCON3.

◆ glcd_tcon_signal_select_t

enum glcd_tcon_signal_select_t	
Timing signals for driving the LCD panel	
Enumerator	
GLCD_TCON_SIGNAL_SELECT_STVA_VS	STVA/VS.
GLCD_TCON_SIGNAL_SELECT_STVB_VE	STVB/VE.
GLCD_TCON_SIGNAL_SELECT_STHA_HS	STH/SP/HS.
GLCD_TCON_SIGNAL_SELECT_STHB_HE	STB/LP/HE.
GLCD_TCON_SIGNAL_SELECT_DE	DE.

Function Documentation

◆ R_GLCD_Close()

ssp_err_t R_GLCD_Close (display_ctrl_t *const *p_api_ctrl*)

Close GLCDC module.

Implements

- display_api_t::close.

Return values

SSP_SUCCESS	Device was closed successfully.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed when the GLCD is updating register values internally.

Note

This API can be called when the driver is not in DISPLAY_STATE_CLOSED state. It returns an error if the register update operation for the background screen generation block is being held.

Disable the GLCD interrupts

Reset the GLCD hardware

Halt the peripheral clock to the GLCD module

Unlock the GLCD resource

◆ R_GLCD_ClutUpdate()

ssp_err_t R_GLCD_ClutUpdate (display_ctrl_t const *const *p_api_ctrl*, display_clut_cfg_t const *const *p_clut_cfg*, display_frame_layer_t *frame*)

Update Color Look Up Table of GLCDC module.

Implements

- display_api_t::clut.

Return values

SSP_SUCCESS	CLUT updated successfully.
SSP_ERR_ASSERTION	Pointer to the control block or CLUT source data is NULL.
SSP_ERR_INVALID_CLUT_ACCESS	Illegal CLUT entry or size is specified.

Note

This API can be called any time.

Check the CLUT table current used

Copy the new CLUT data on the source memory to the CLUT SRAM in the GLCD module

Make the GLCD module read the new CLUT table data from the next frame

◆ R_GLCD_ColorCorrection()

ssp_err_t R_GLCD_ColorCorrection (display_ctrl_t const *const *p_api_ctrl*, display_correction_t const *const *p_correction*)

Perform color correction by GLCDC module.

Implements

- display_api_t::correction.

Return values

SSP_SUCCESS	Color correction by GLCDC module was performed successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the display correction structure is NULL.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCDC is updating registers internally.

Note

This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the output control block is being held.

Configure the brightness and contrast correction register setting.

Update the Output block register setting.

◆ R_GLCD_LayerChange()

ssp_err_t R_GLCD_LayerChange (display_ctrl_t const *const *p_api_ctrl*, display_runtime_cfg_t const *const *p_cfg*, display_frame_layer_t *frame*)

Change layer parameters of GLCDC module at runtime.

Implements

- display_api_t::layerChange.

Return values

SSP_SUCCESS	Changed layer parameters of GLCDC module successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_MODE	A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_ARGUMENT	An invalid parameter is found in the argument.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCD is updating register values internally.

Note

This API can be called when the driver is in DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.

Configure the graphics plane layers

Reflect the graphics module register value to the GLCD internal operations (at the timing of the next Vsync assertion)

◆ R_GLCD_Open()

ssp_err_t R_GLCD_Open (display_ctrl_t *const *p_api_ctrl*, display_cfg_t const *const *p_cfg*)

Open GLCDC module.

Implements

- display_api_t::open.

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_HW_LOCKED	GLCDC resource is locked.
SSP_ERR_CLOCK_GENERATION	Dot clock cannot be generated from clock

	source.
SSP_ERR_INVALID_TIMING_SETTING	Invalid panel timing parameter.
SSP_ERR_INVALID_LAYER_SETTING	Invalid layer setting found.
SSP_ERR_INVALID_LAYER_FORMAT	Invalid format is specified.
SSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction setting found.

Note

PCLKA must be supplied to Graphics LCD Controller (GLCDC) and GLCDC pins must be set in IOPORT before calling this API.

Lock the GLCD resource

Supply the peripheral clock to the GLCD module

Release GLCD from a SW reset status.

Set the dot clock frequency

Set the panel signal timing

Configure the background screen

Store back poach position to the control block (needed to define the layer blending position later)

Configure the graphics plane layers

Configure the output control block

Configure the color correction setting (brightness, brightness and gamma correction)

Change GLCD driver state

Save callback function

Save user defined context

Save the display interface context into GLCD HAL control block

Set the line number which is suppose to happen the line detect interrupt

◆ R_GLCD_Start()

ssp_err_t R_GLCD_Start (display_ctrl_t *const *p_api_ctrl*)

Start GLCDC module.

Implements

- display_api_t::start.

Return values

SSP_SUCCESS	Device was started successfully.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_OPENED.

Note

This API can be called when the driver is not in DISPLAY_STATE_OPENED status.

Start to output the vertical and horizontal synchronization signals and screen data.

Enable Line detect function

◆ R_GLCD_StatusGet()

ssp_err_t R_GLCD_StatusGet (display_ctrl_t const *const *p_api_ctrl*, display_status_t *const *p_status*)

Get status of GLCDC module.

Implements

- display_api_t::statusGet.

Return values

SSP_SUCCESS	Got status successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the status structure is NULL.

Note

The GLCD hardware starts the fading processing at the first Vsync after the previous LayerChange() call is held. Due to this behavior of the hardware, this API may not return DISPLAY_FADE_STATUS_FADING_UNDERWAY as the fading status, if it is called before the first Vsync after LayerChange() is called. In this case, the API returns DISPLAY_FADE_STATUS_UNCERTAIN, instead of DISPLAY_FADE_STATUS_NOT_UNDERWAY.

Return the GLCD HAL driver state

Return the fading status for the layers

◆ R_GLCD_Stop()

ssp_err_t R_GLCD_Stop (display_ctrl_t *const *p_api_ctrl*)

Stop GLCDC module.

Implements

- display_api_t::stop.

Return values

SSP_SUCCESS	Device was stopped successfully
SSP_ERR_ASSERTION	Pointer to the control block is NULL
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_UPDATE_TIMING	The function call is performed while the GLCD is updating register values internally.

Note

This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks, the graphics data I/F blocks, or the output control block is being held.

Stop outputting the vertical and horizontal synchronization signals and screen data.

◆ R_GLCD_VersionGet()

ssp_err_t R_GLCD_VersionGet (ssp_version_t * *p_version*)

Get version of R_GLCDC module.

Implements

- display_api_t::versionGet.

Parameters

p_version	The version number

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	NULL pointer is passed to function.

Note

This function is re-entrant.

glcd_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GLCDC

#include <r_glcd.h>

Data Fields

display_state_t	state Status of GLCD module.
void(*	p_callback)(display_callback_args_t *p_args) Pointer to callback function.
void const *	p_context Pointer to the higher level device context.
R_GLCDC_Type *	p_reg Base register address.

Detailed Description

Display control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_glcd.h

glcd_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GLCDC

#include <r_glcd.h>

Data Fields

glcd_tcon_pin_t	tcon_hsync GLCD TCON output pin select.
glcd_tcon_pin_t	tcon_vsync

	GLCD TCON output pin select.
glcd_tcon_pin_t	tcon_de GLCD TCON output pin select.
glcd_correction_proc_order_t	correction_proc_order Correction control route select.
glcd_clk_src_t	clksrc Clock Source selection.
glcd_panel_clk_div_t	clock_div_ratio Clock divide ratio for dot clock.
glcd_dithering_mode_t	dithering_mode Dithering mode.
glcd_dithering_pattern_t	dithering_pattern_A Dithering pattern A.
glcd_dithering_pattern_t	dithering_pattern_B Dithering pattern B.
glcd_dithering_pattern_t	dithering_pattern_C Dithering pattern C.
glcd_dithering_pattern_t	dithering_pattern_D Dithering pattern D.

Detailed Description

GLCD hardware specific configuration

The documentation for this struct was generated from the following file:

- r_glcd.h

glcd_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GLCDC

#include <r_glcd.h>

Data Fields

display_coordinate_t	back_porch Zero coordinate for graphics plane(Bach porch End)
uint16_t	hsize Horizontal pixel size in a line.
uint16_t	vsize Vertical pixel size in a frame.
bsp_lock_t	resource_lock Resource lock.
void *	p_context

Detailed Description

GLCD hardware specific control block

Field Documentation

◆ p_context

void* glcd_ctrl_t::p_context
Pointer to the function level device context (e.g. display_ctrl_t type data)

The documentation for this struct was generated from the following file:

- r_glcd.h

5.1.5.22 GPT

Renesas Synergy Software Package Reference » HAL Layer

Driver for the General PWM Timer (GPT). More...

Data Structures

struct	gpt_output_pin_t
struct	gpt_instance_ctrl_t
struct	timer_on_gpt_cfg_t

Enumerations

enum	gpt_pin_level_t { GPT_PIN_LEVEL_LOW = 0, GPT_PIN_LEVEL_HIGH = 1, GPT_PIN_LEVEL_RETAINED = 2 }
enum	gpt_shortest_level_t { GPT_SHORTEST_LEVEL_OFF = 0, GPT_SHORTEST_LEVEL_ON = 1 }
enum	gpt_trigger_t { GPT_TRIGGER_NONE = 0, GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_LOW = (1UL << 8), GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_HIGH = (1UL << 9), GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_LOW = (1UL << 10), GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_HIGH = (1UL << 11), GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_LOW = (1UL << 12), GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_HIGH = (1UL << 13), GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_LOW = (1UL << 14), GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_HIGH = (1UL << 15), GPT_TRIGGER_SOURCE_REGISTER_ENABLE = (1UL << 31) }
enum	gpt_output_t { GPT_OUTPUT_RETAINED = 0, GPT_OUTPUT_LOW = 1, GPT_OUTPUT_HIGH = 2, GPT_OUTPUT_TOGGLED = 3 }

Functions

ssp_err_t	R_GPT_TimerOpen (timer_ctrl_t *const p_api_ctrl, timer_cfg_t const *const p_cfg) Powers on GPT, handles required initialization described in hardware

	manual. Implements timer_api_t::open. More...
ssp_err_t	R_GPT_Stop (timer_ctrl_t *const p_api_ctrl) Stops timer. Implements timer_api_t::stop. More...
ssp_err_t	R_GPT_Start (timer_ctrl_t *const p_api_ctrl) Starts timer. Implements timer_api_t::start. More...
ssp_err_t	R_GPT_CounterGet (timer_ctrl_t *const p_api_ctrl, timer_size_t *const p_value) Sets counter value in provided p_value pointer. Implements timer_api_t::counterGet. More...
ssp_err_t	R_GPT_Reset (timer_ctrl_t *const p_api_ctrl) Resets the counter value to 0. Implements timer_api_t::reset. More...
ssp_err_t	R_GPT_PeriodSet (timer_ctrl_t *const p_api_ctrl, timer_size_t const period, timer_unit_t const unit) Sets period value provided. Implements timer_api_t::periodSet. More...
ssp_err_t	R_GPT_DutyCycleSet (timer_ctrl_t *const p_api_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const unit, uint8_t const pin) Sets status in provided p_status pointer. Implements pwm_api_t::dutyCycleSet. More...
ssp_err_t	R_GPT_InfoGet (timer_ctrl_t *const p_api_ctrl, timer_info_t *const p_info) Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet. More...
ssp_err_t	R_GPT_Close (timer_ctrl_t *const p_api_ctrl) Stops counter, disables output pins, and clears internal driver data. More...
ssp_err_t	R_GPT_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...

Detailed Description

Driver for the General PWM Timer (GPT).

Summary

Extends Timer Interface.

This module implements the Timer Interface using the General PWM Timer (GPT) peripherals GPT32EH, GPT32E, GPT32. It also provides an output compare extension to output the timer signal to the GTIOC pin.

Enumeration Type Documentation

◆ gpt_output_t

enum gpt_output_t	
Output level used when selecting what happens at compare match or cycle end.	
Enumerator	
GPT_OUTPUT_RETAINED	Output retained.
GPT_OUTPUT_LOW	Output low.
GPT_OUTPUT_HIGH	Output high.
GPT_OUTPUT_TOGGLED	Output toggled.

◆ gpt_pin_level_t

enum gpt_pin_level_t	
Level of GPT pin	
Enumerator	
GPT_PIN_LEVEL_LOW	Pin level low.
GPT_PIN_LEVEL_HIGH	Pin level high.
GPT_PIN_LEVEL_RETAINED	Pin level retained.

◆ gpt_shortest_level_t

enum gpt_shortest_level_t	
GPT PWM shortest pin level	
Enumerator	
GPT_SHORTEST_LEVEL_OFF	1 extra PCLK in ON time. Minimum ON time will be limited to 2 PCLK raw counts.
GPT_SHORTEST_LEVEL_ON	1 extra PCLK in OFF time. Minimum ON time will be limited to 1 PCLK raw counts.

◆ gpt_trigger_t

enum gpt_trigger_t	
Sources can be used to start the timer, stop the timer, count up, or count down.	
Enumerator	
GPT_TRIGGER_NONE	No action performed.
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input rises while GTIOCB is low.
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input rises while GTIOCB is high.
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input falls while GTIOCB is low.
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input falls while GTIOCB is high.
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input rises while GTIOCA is low.
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input rises while GTIOCA is high.
GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input falls while GTIOCA is low.
GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input falls while GTIOCA is high.
GPT_TRIGGER_SOURCE_REGISTER_ENABLE	Enables settings in the Source Select Register.

Function Documentation

◆ R_GPT_Close()

ssp_err_t R_GPT_Close (timer_ctrl_t *const *p_api_ctrl*)

Stops counter, disables output pins, and clears internal driver data.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Cleanup. Stop counter and disable output.

Unlock channel

Clear stored internal driver data

◆ R_GPT_CounterGet()

ssp_err_t R_GPT_CounterGet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t *const *p_value*)

Sets counter value in provided p_value pointer. Implements timer_api_t::counterGet.

Return values

SSP_SUCCESS	Counter value read, p_value is valid.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Read counter value

◆ R_GPT_DutyCycleSet()

ssp_err_t R_GPT_DutyCycleSet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t const *duty_cycle*, timer_pwm_unit_t const *unit*, uint8_t const *pin*)

Sets status in provided p_status pointer. Implements pwm_api_t::dutyCycleSet.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INVALID_ARGUMENT	The pin value is out of range; Should be either 0 (for GTIOCA) or 1 (for GTIOCB).

Converted duty cycle to PCLK counts before it can be set in registers

Set duty cycle.

◆ R_GPT_InfoGet()

ssp_err_t R_GPT_InfoGet (timer_ctrl_t *const *p_api_ctrl*, timer_info_t *const *p_info*)

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Return values

SSP_SUCCESS	Period, count direction, frequency, and status value written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_info parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Get and store period

Get and store clock frequency

Get and store clock counting direction

◆ R_GPT_PeriodSet()

ssp_err_t R_GPT_PeriodSet (timer_ctrl_t *const *p_api_ctrl*, timer_size_t const *period*, timer_unit_t const *unit*)

Sets period value provided. Implements timer_api_t::periodSet.

Return values

SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARGUMENT	One of the following is invalid: • p_period->unit: must be one of the options from timer_unit_t • p_period->value: must result in a period in the following range: ◦ Lower bound: (1 / (PCLK frequency)) ◦ Upper bound: (0xFFFFFFFF * 1024 / (PCLK frequency))
SSP_ERR_NOT_OPEN	The channel is not opened.

Delay must be converted to PCLK counts before it can be set in registers

Make sure period is valid.

Store current status, then stop timer before setting divisor register

Reset counter in case new cycle is less than current count value, then restore state (counting or stopped).

◆ R_GPT_Reset()

ssp_err_t R_GPT_Reset (timer_ctrl_t *const *p_api_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Return values

SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Write the counter value

◆ R_GPT_Start()

ssp_err_t R_GPT_Start (timer_ctrl_t *const *p_api_ctrl*)

Starts timer. Implements timer_api_t::start.

Return values

SSP_SUCCESS	Timer successfully started.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Start timer

◆ R_GPT_Stop()

ssp_err_t R_GPT_Stop (timer_ctrl_t *const *p_api_ctrl*)

Stops timer. Implements timer_api_t::stop.

Return values

SSP_SUCCESS	Timer successfully stopped.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Stop timer

◆ R_GPT_TimerOpen()

ssp_err_t R_GPT_TimerOpen (timer_ctrl_t *const *p_api_ctrl*, timer_cfg_t const *const *p_cfg*)

Powers on GPT, handles required initialization described in hardware manual. Implements timer_api_t::open.

The Open function configures a single GPT channel, starts the channel, and provides a handle for use with the GPT API Control and Close functions. This function must be called once prior to calling any other GPT API functions. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

GPT hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The GPT implementation of the general timer can accept a timer_on_gpt_cfg_t extension parameter.

Return values

SSP_SUCCESS	Initialization was successful and timer has started.

SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_cfg is NULL, OR • p_ctrl is NULL, OR
SSP_ERR_INVALID_ARGUMENT	One of the following parameters is invalid: • p_cfg->period: must be in the following range: ◦ Lower bound: (1 / (PCLK frequency) ◦ Upper bound: (0xFFFFFFFF * 1024 / (PCLK frequency)) • p_cfg->p_callback not NULL, but ISR is not enabled. ISR must be enabled to use callback function. Enable channel's overflow ISR in bsp_irq_cfg.h.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_IRQ_BSP_DISABLED	- p_cfg->mode is TIMER_MODE_ONE_SHOT, but ISR is not enabled. ISR must be enabled to use one-shot mode.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	- The channel requested in the p_cfg parameter is not available on this device.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Calculate period and store internal variables

Save the configuration

Calculate duty cycle

Verify channel is not already used

Power on GPT before setting any hardware registers. Make sure the counter is stopped before setting mode register, PCLK divisor register, and counter register.

◆ R_GPT_VersionGet()

ssp_err_t R_GPT_VersionGet (ssp_version_t *const *p_version*)
Sets driver version based on compile time macros. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — The parameter p_version is NULL.

gpt_output_pin_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GPT

#include <r_gpt.h>

Data Fields

bool	output_enabled Set to true to enable output, false to disable output.
gpt_pin_level_t	stop_level Select a stop level from gpt_pin_level_t.

Detailed Description

Configurations for output pins.

The documentation for this struct was generated from the following file:

- r_gpt.h

gpt_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GPT

#include <r_gpt.h>

Data Fields

void(*	p_callback)(timer_callback_args_t *p_args)
void const *	p_context
void *	p_reg Base register for this channel.
uint32_t	open Whether or not channel is open.
uint8_t	channel Channel number.
bool	one_shot Whether or not timer is in one shot mode.
bool	gtioca_output_enabled Set to true to enable gtioca pin output.
bool	gtiocb_output_enabled Set to true to enable gtiocb pin output.
IRQn_Type	irq Counter overflow IRQ number.
timer_variant_t	variant Timer variant.
gpt_shortest_level_t	shortest_pwm_signal Shortest PWM signal level.

Detailed Description

Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

Field Documentation

◆ p_callback

void(* gpt_instance_ctrl_t::p_callback) (timer_callback_args_t *p_args)
Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.

◆ p_context

void const* gpt_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in timer_callback_args_t.

The documentation for this struct was generated from the following file:

- r_gpt.h

timer_on_gpt_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GPT

#include <r_gpt.h>

Data Fields

gpt_output_pin_t	gtioca Configuration for GPT I/O pin A.
gpt_output_pin_t	gtiocb Configuration for GPT I/O pin B.
gpt_shortest_level_t	shortest_pwm_signal Shortest PWM signal level.

Detailed Description

GPT extension configures the output pins for GPT.

The documentation for this struct was generated from the following file:

- r_gpt.h

5.1.5.23 GPT Input Capture

Renesas Synergy Software Package Reference » HAL Layer

Driver for the General PWM Timer (GPT) with Input Capture. More...

Data Structures

struct	gpt_input_capture_extend_t Extension configuration struct for TU Input Capture. More...
struct	gpt_input_capture_instance_ctrl_t

Macros

#define	GPT_INPUT_CAPTURE_CODE_VERSION_MAJOR (2U)
#define	GPT_INPUT_CAPTURE_MAX_COUNT (0xFFFFFFFFUL) Maximum value of GPT counter.

Enumerations

enum	gpt_input_capture_signal_t { GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCA, GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCB }
enum	gpt_input_capture_signal_filter_t { GPT_INPUT_CAPTURE_SIGNAL_FILTER_1, GPT_INPUT_CAPTURE_SIGNAL_FILTER_4, GPT_INPUT_CAPTURE_SIGNAL_FILTER_16, GPT_INPUT_CAPTURE_SIGNAL_FILTER_64 }

enum	gpt_input_capture_clock_divider_t {
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1,
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_4,
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_16,
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_64,
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_256,
	GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1024
	}

Functions

ssp_err_t	R_GPT_InputCaptureOpen (input_capture_ctrl_t *const p_api_ctrl, input_capture_cfg_t const *const p_cfg) Open a GPT Timer for Input Capture. Implements input_capture_api_t::open. More...
ssp_err_t	R_GPT_InputCaptureClose (input_capture_ctrl_t *const p_api_ctrl) Close a GPT Timer Channel for Input Capture. Implements input_capture_api_t::close. More...
ssp_err_t	R_GPT_InputCaptureVersionGet (ssp_version_t *const p_version) Gets driver version based on compile time macros. Implements input_capture_api_t::versionGet. More...
ssp_err_t	R_GPT_InputCaptureDisable (input_capture_ctrl_t const *const p_api_ctrl) Disables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements input_capture_api_t::disable. More...
ssp_err_t	R_GPT_InputCaptureEnable (input_capture_ctrl_t const *const p_api_ctrl) Enables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements input_capture_api_t::enable. More...
ssp_err_t	R_GPT_InputCaptureInfoGet (input_capture_ctrl_t const *const p_api_ctrl, input_capture_info_t *const p_info) Gets status into provided p_info pointer. Implements input_capture_api_t::infoGet. More...
ssp_err_t	R_GPT_InputCaptureLastCaptureGet (input_capture_ctrl_t const *const p_api_ctrl, input_capture_capture_t *const p_capture) Update the last captured value and overflow count, in provided p_capture pointer. Implements input_capture_api_t::lastCaptureGet. More...

Detailed Description

Driver for the General PWM Timer (GPT) with Input Capture.

Summary

Extends Input Capture Interface.

This module implements the Input Capture Interface for the General PWM Timer (GPT) peripherals GPT32EH, GPT32E, GPT32.

Macro Definition Documentation

◆ GPT_INPUT_CAPTURE_CODE_VERSION_MAJOR

#define GPT_INPUT_CAPTURE_CODE_VERSION_MAJOR (2U)
Includes

Enumeration Type Documentation

◆ gpt_input_capture_clock_divider_t

enum gpt_input_capture_clock_divider_t	
Input capture PCLK divider. Used to scale the timer counter.	
Enumerator	
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1	/ 1
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_4	/ 4
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_16	/ 16
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_64	/ 64
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_256	/ 256
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1024	/ 1024

◆ gpt_input_capture_signal_filter_t

enum gpt_input_capture_signal_filter_t	
Input capture signal noise filter (debounce) setting. Only available for input signals GTIOCxA and GTIOCxB. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, GPT section.	
Enumerator	
GPT_INPUT_CAPTURE_SIGNAL_FILTER_1	PCLK/1 - fast sampling.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_4	PCLK/4.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_16	PCLK/16.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_64	PCLK/64 - slow sampling.

◆ gpt_input_capture_signal_t

enum gpt_input_capture_signal_t	
Input capture signal selection	
Enumerator	
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCA	GTIOCxA pin, where x is channel number.
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCB	GTIOCxB pin, where x is channel number.

Function Documentation

◆ R_GPT_InputCaptureClose()

ssp_err_t R_GPT_InputCaptureClose (input_capture_ctrl_t *const *p_api_ctrl*)

Close a GPT Timer Channel for Input Capture. Implements input_capture_api_t::close.

Clears Timer settings, disables interrupts, and clears internal driver data.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Cleanup. Disable interrupts and stop measurements.

Unlock channel

Clear stored internal driver data

◆ R_GPT_InputCaptureDisable()

ssp_err_t R_GPT_InputCaptureDisable (input_capture_ctrl_t const *const *p_api_ctrl*)

Disables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements input_capture_api_t::disable.

Return values

SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Disable interrupts

Clearing the input capture source select registers.

◆ R_GPT_InputCaptureEnable()

ssp_err_t R_GPT_InputCaptureEnable (input_capture_ctrl_t const *const *p_api_ctrl*)

Enables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements input_capture_api_t::enable.

Return values

SSP_SUCCESS	Interrupt enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Configuring the input capture source select registers.

Enabling the overflow and capture registers.

◆ R_GPT_InputCaptureInfoGet()

ssp_err_t R_GPT_InputCaptureInfoGet (input_capture_ctrl_t const *const *p_api_ctrl*, input_capture_info_t *const *p_info*)

Gets status into provided p_info pointer. Implements input_capture_api_t::infoGet.

Return values

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_GPT_InputCaptureLastCaptureGet()

ssp_err_t R_GPT_InputCaptureLastCaptureGet (input_capture_ctrl_t const *const *p_api_ctrl*, input_capture_capture_t *const *p_capture*)

Update the last captured value and overflow count, in provided p_capture pointer. Implements input_capture_api_t::lastCaptureGet.

Return values

SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Set capture value

◆ R_GPT_InputCaptureOpen()

ssp_err_t R_GPT_InputCaptureOpen (input_capture_ctrl_t *const *p_api_ctrl*, input_capture_cfg_t const *const *p_cfg*)

Open a GPT Timer for Input Capture. Implements input_capture_api_t::open.

The Open function configures a single GPT channel for input capture and provides a handle for use with the other Input Capture API functions. This function must be called once prior to calling any other Input Capture API function. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the parameters is NULL: p_cfg, p_ctrl, p_extend.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Verify channel is not already used

◆ R_GPT_InputCaptureVersionGet()

ssp_err_t R_GPT_InputCaptureVersionGet (ssp_version_t *const *p_version*)

Gets driver version based on compile time macros. Implements input_capture_api_t::versionGet.

Return values

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

gpt_input_capture_extend_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GPT Input Capture

Extension configuration struct for TU Input Capture. More...

#include <r_gpt_input_capture.h>

Data Fields

gpt_input_capture_signal_t	signal One of gpt_input_capture_signal_t.
gpt_input_capture_signal_filter_t	signal_filter One of gpt_input_capture_signal_filter_t.
gpt_input_capture_clock_divider_t	clock_divider One of gpt_input_capture_clock_divider_t.
input_capture_signal_level_t	enable_level
bool	enable_filter One of gpt_input_capture_signal_filter_t.

Detailed Description

Extension configuration struct for TU Input Capture.

Pointed to by input_capture_cfg_t.p_extend

Field Documentation

◆ enable_level

input_capture_signal_level_t gpt_input_capture_extend_t::enable_level
The unused GTIOCx pin can be used as an enable signal to enable captures. If the Input Capture Signal Pin is GTIOCA, then the enable pin is GTIOCB. The enable level is set here if used.

The documentation for this struct was generated from the following file:

- r_gpt_input_capture.h

gpt_input_capture_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » GPT Input Capture

#include <r_gpt_input_capture.h>

Data Fields

uint32_t	open Whether or not channel is open.
uint8_t	channel The channel in use.
input_capture_mode_t	mode The mode of measurement being performed.
input_capture_repetition_t	repetition One-shot or periodic measurement.
uint32_t	capture_count The value of the timer captured at the time of interrupt.
uint32_t	overflows_last Overflow count that occurred during last measurement.
uint32_t	overflows_current Running count of overflows in current measurement.
void(*	p_callback)(input_capture_callback_args_t *p_args) Pointer to user callback.
void const *	p_context

	Pointer to user's context data, to be passed to the callback function.
void *	p_reg GPT base register for this channel.
IRQn_Type	capture_irq Capture IRQ number.
IRQn_Type	overflow_irq Overflow IRQ number.
input_capture_variant_t	variant Timer variant.
uint32_t	start_bitmask Start and Clear bitmask for input capture.
uint32_t	stop_bitmask Stop and capture bitmask for input capture.

Detailed Description

Channel control block. DO NOT INITIALIZE. Initialization occurs when input_capture_api_t::open is called.

The documentation for this struct was generated from the following file:

- r_gpt_input_capture.h

5.1.5.24 ICU

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Interrupt Controller Unit (ICU) External pin interrupts function. More...

Data Structures

struct	icu_instance_ctrl_t

Functions

ssp_err_t	R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const p_api_ctrl, external_irq_cfg_t const *const p_cfg) Configure an external input pin for use with the button interface. Implements external_irq_api_t::open. More...
ssp_err_t	R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const p_api_ctrl) Enable external interrupt for specified channel at NVIC. Implements external_irq_api_t::enable. More...
ssp_err_t	R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const p_api_ctrl) Disable external interrupt for specified channel at NVIC. Implements external_irq_api_t::disable. More...
ssp_err_t	R_ICU_ExternalIrqTriggerSet (external_irq_ctrl_t *const p_api_ctrl, external_irq_trigger_t hw_trigger) Set trigger value provided. Implements external_irq_api_t::triggerSet. More...
ssp_err_t	R_ICU_ExternalIrqFilterEnable (external_irq_ctrl_t *const p_api_ctrl) Enable external interrupt digital filter for specified channel. Implements external_irq_api_t::filterEnable. More...
ssp_err_t	R_ICU_ExternalIrqFilterDisable (external_irq_ctrl_t *const p_api_ctrl) Enable external interrupt digital filter for specified channel. Implements external_irq_api_t::filterDisable. More...
ssp_err_t	R_ICU_ExternalIrqVersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. Implements external_irq_api_t::versionGet. More...
ssp_err_t	R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_api_ctrl) Disable external interrupt. Implements external_irq_api_t::close. More...

Detailed Description

Driver for the Interrupt Controller Unit (ICU) External pin interrupts function.

Summary

Extends External IRQ Interface.

This module implements the External IRQ Interface using the external input pins in the Interrupt Controller Unit (ICU).

Function Documentation

◆ R_ICU_ExternalIrqClose()

ssp_err_t R_ICU_ExternalIrqClose (external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt. Implements external_irq_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Cleanup. Disable interrupt

Disable the interrupt, and then clear the interrupt pending bits and interrupt status.

Release BSP hardware lock

◆ R_ICU_ExternalIrqDisable()

ssp_err_t R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt for specified channel at NVIC. Implements external_irq_api_t::disable.

Return values

SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	Requested IRQ is not defined in this system

Disable the interrupt, and then clear the interrupt pending bits and interrupt status.

◆ R_ICU_ExternalIrqEnable()

ssp_err_t R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt for specified channel at NVIC. Implements external_irq_api_t::enable.

Return values

SSP_SUCCESS	Interrupt Enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	Requested IRQ is not defined in this system

Clear the interrupt status and Pending bits, before the interrupt is enabled.

◆ R_ICU_ExternalIrqFilterDisable()

ssp_err_t R_ICU_ExternalIrqFilterDisable (external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt digital filter for specified channel. Implements external_irq_api_t::filterDisable.

Return values

SSP_SUCCESS	External interrupt digital filter disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Disable external interrupt digital filter

◆ R_ICU_ExternalIrqFilterEnable()

ssp_err_t R_ICU_ExternalIrqFilterEnable (external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt digital filter for specified channel. Implements external_irq_api_t::filterEnable.

Return values

SSP_SUCCESS	External interrupt digital filter enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Enable external interrupt digital filter

◆ R_ICU_ExternalIrqOpen()

ssp_err_t R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const *p_api_ctrl*, external_irq_cfg_t const *const *p_cfg*)

Configure an external input pin for use with the button interface. Implements external_irq_api_t::open.

The Open function is responsible for preparing an external input pin for operation. After completion of the Open function the external input pin shall be enabled and ready to service interrupts. This function must be called once prior to calling any other external input pin API functions. Once successfully completed, the status of the selected external input pin will be set to "open". After that this function should not be called again for the same external input pin without first performing a "close" by calling R_ICU_ExternalIrqClose().

Return values

SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	One of the following is invalid: • p_ctrl or p_cfg is NULL • The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h.
SSP_ERR_INVALID_ARGUMENT	p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. Enable channel's overflow ISR in bsp_irq_cfg.h.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Get fmi feature information for ICU.

Acquire hardware lock for this specific external IRQ channel of the ICU

Initialize control block.

Configure interrupt if enabled

Perform hardware initializations based on configuration.

Mark the control block as open

◆ R_ICU_ExternalIrqTriggerSet()

ssp_err_t R_ICU_ExternalIrqTriggerSet (external_irq_ctrl_t *const *p_api_ctrl*, external_irq_trigger_t *hw_trigger*)

Set trigger value provided. Implements external_irq_api_t::triggerSet.

Return values

SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_period parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Set trigger value provided

◆ R_ICU_ExternalIrqVersionGet()

ssp_err_t R_ICU_ExternalIrqVersionGet (ssp_version_t *const *p_version*)

Set driver version based on compile time macros. Implements external_irq_api_t::versionGet.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Read the driver version

icu_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » ICU

#include <r_icu.h>

Data Fields

uint32_t	open Used to determine if channel control block is in use.
R_ICU_Type *	p_reg Pointer to register base address.

void(*	p_callback)(external_irq_callback_args_t *p_args)
void const *	p_context
IRQn_Type	irq NVIC interrupt number.
uint8_t	channel Channel.

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in external_irq_api_t::open.

Field Documentation

◆ p_callback

void(* icu_instance_ctrl_t::p_callback) (external_irq_callback_args_t *p_args)
Callback provided when a external IRQ ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void const* icu_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

The documentation for this struct was generated from the following file:

- r_icu.h

5.1.5.25 IOPORT

Renesas Synergy Software Package Reference » HAL Layer

Driver for the I/O Ports. More...

Functions

ssp_err_t	R_IOPORT_Init (const ioport_cfg_t *p_cfg)

	Initializes internal driver data, then calls R_IOPORT_PinsCfg to configure pins. More...
ssp_err_t	R_IOPORT_PinsCfg (const ioport_cfg_t *p_cfg) Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg. More...
ssp_err_t	R_IOPORT_PinCfg (ioport_port_pin_t pin, uint32_t cfg) Configures the settings of a pin. Implements ioport_api_t::pinCfg. More...
ssp_err_t	R_IOPORT_PinRead (ioport_port_pin_t pin, ioport_level_t *p_pin_value) Reads the level on a pin. Implements ioport_api_t::pinRead. More...
ssp_err_t	R_IOPORT_PortRead (ioport_port_t port, ioport_size_t *p_port_value) Reads the value on an IO port. Implements ioport_api_t::portRead. More...
ssp_err_t	R_IOPORT_PortWrite (ioport_port_t port, ioport_size_t value, ioport_size_t mask) Writes to multiple pins on a port. Implements ioport_api_t::portWrite. More...
ssp_err_t	R_IOPORT_PinWrite (ioport_port_pin_t pin, ioport_level_t level) Sets a pin's output either high or low. Implements ioport_api_t::pinWrite. More...
ssp_err_t	R_IOPORT_PortDirectionSet (ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask) Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet(). More...
ssp_err_t	R_IOPORT_PinDirectionSet (ioport_port_pin_t pin, ioport_direction_t direction) Sets the direction of an individual pin on a port. Implements ioport_api_t::pinDirectionSet. More...

ssp_err_t	R_IOPORT_PortEventInputRead (ioport_port_t port, ioport_size_t *p_event_data) Reads the value of the event input data. Implements ioport_api_t::portEventInputRead(). More...
ssp_err_t	R_IOPORT_PinEventInputRead (ioport_port_pin_t pin, ioport_level_t *p_pin_event) Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead. More...
ssp_err_t	R_IOPORT_PortEventOutputWrite (ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value) This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite. More...
ssp_err_t	R_IOPORT_PinEventOutputWrite (ioport_port_pin_t pin, ioport_level_t pin_value) This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite. More...
ssp_err_t	R_IOPORT_VersionGet (ssp_version_t *p_data) Returns IOPort HAL driver version. Implements ioport_api_t::versionGet. More...
ssp_err_t	R_IOPORT_EthernetModeCfg (ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode) Configures Ethernet channel PHY mode. Implements ioport_api_t::ethModeCfg. More...

Detailed Description

Driver for the I/O Ports.

The IOPort HAL drivers provide the ability to access the I/O Ports of a device at both bit and port level. Port and pin direction can be changed. In addition a number of configuration APIs are provided to change the functionality of individual pins.

Function Documentation

◆ R_IOPORT_EthernetModeCfg()

ssp_err_t R_IOPORT_EthernetModeCfg (ioport_ethernet_channel_t *channel*, ioport_ethernet_mode_t *mode*)

Configures Ethernet channel PHY mode. Implements ioport_api_t::ethModeCfg.

Return values

SSP_SUCCESS	Ethernet PHY mode set.
SSP_ERR_INVALID_ARGUMENT	Channel or mode not valid.
SSP_ERR_UNSUPPORTED	Ethernet configuration not supported on this device.

Note

This function is not re-entrant.

◆ R_IOPORT_Init()

ssp_err_t R_IOPORT_Init (const ioport_cfg_t * *p_cfg*)

Initializes internal driver data, then calls R_IOPORT_PinsCfg to configure pins.

Return values

SSP_SUCCESS	Pin configuration data written to PFS register(s)
SSP_ERR_ASSERTION	NULL pointer

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

◆ R_IOPORT_PinCfg()

ssp_err_t R_IOPORT_PinCfg (ioport_port_pin_t *pin*, uint32_t *cfg*)

Configures the settings of a pin. Implements ioport_api_t::pinCfg.

Return values

SSP_SUCCESS	Pin configured.
SSP_ERR_INVALID_ARGUMENT	Invalid pin

Note

This function is re-entrant for different pins. This function will change the configuration of the pin with the new configuration. For example it is not possible with this function to change the drive strength of a pin while leaving all the other pin settings unchanged. To achieve this the original settings with the required change will need to be written using this function.

◆ R_IOPORT_PinDirectionSet()

ssp_err_t R_IOPORT_PinDirectionSet (ioport_port_pin_t *pin*, ioport_direction_t *direction*)

Sets the direction of an individual pin on a port. Implements ioport_api_t::pinDirectionSet.

Return values

SSP_SUCCESS	Pin direction updated.
SSP_ERR_INVALID_ARGUMENT	The pin and/or direction not valid.

Note

This function is re-entrant for different pins.

◆ R_IOPORT_PinEventInputRead()

ssp_err_t R_IOPORT_PinEventInputRead (ioport_port_pin_t *pin*, ioport_level_t * *p_pin_event*)

Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead.

The pin event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

SSP_SUCCESS	Pin read.
SSP_ERR_INVALID_ARGUMENT	Pin not valid.
SSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant.

◆ R_IOPORT_PinEventOutputWrite()

ssp_err_t R_IOPORT_PinEventOutputWrite (ioport_port_pin_t *pin*, ioport_level_t *pin_value*)

This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite.

Using the event system enables a pin state to be stored by this function in advance of being output on the pin. The output to the pin will occur when the ELC event occurs.

Return values

SSP_SUCCESS	Pin event data written.
SSP_ERR_INVALID_ARGUMENT	Pin or value not valid.

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PinRead()

ssp_err_t R_IOPORT_PinRead (ioport_port_pin_t *pin*, ioport_level_t * *p_pin_value*)

Reads the level on a pin. Implements ioport_api_t::pinRead.

Return values

SSP_SUCCESS	Pin read.
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different pins.

◆ R_IOPORT_PinsCfg()

ssp_err_t R_IOPORT_PinsCfg (const ioport_cfg_t * *p_cfg*)

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg.

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the ISDE pin configurator or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and test.*

Return values

SSP_SUCCESS	Pin configuration data written to PFS register(s)
SSP_ERR_ASSERTION	NULL pointer

◆ R_IOPORT_PinWrite()

ssp_err_t R_IOPORT_PinWrite (ioport_port_pin_t *pin*, ioport_level_t *level*)

Sets a pin's output either high or low. Implements ioport_api_t::pinWrite.

Return values

SSP_SUCCESS	Pin written to.
SSP_ERR_INVALID_ARGUMENT	The pin and/or level not valid.

Note

This function is re-entrant for different pins. This function makes use of the PCNTR3 register to atomically modify the level on the specified pin on a port.

◆ R_IOPORT_PortDirectionSet()

ssp_err_t R_IOPORT_PortDirectionSet (ioport_port_t *port*, ioport_size_t *direction_values*, ioport_size_t *mask*)

Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet().

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. If a bit is set to 1 then the corresponding pin will be changed to an input or an output as specified by the direction values. If a mask bit is set to 0 then the direction of the pin will not be changed.

Return values

SSP_SUCCESS	Port direction updated.
SSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid.

Note

This function is re-entrant for different ports.

High bits

Low bits

New value to write to port direction register

◆ R_IOPORT_PortEventInputRead()

ssp_err_t R_IOPORT_PortEventInputRead (ioport_port_t *port*, ioport_size_t * *p_event_data*)

Reads the value of the event input data. Implements ioport_api_t::portEventInputRead().

The event input data for the port will be read. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

The port event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

SSP_SUCCESS	Port read.
SSP_ERR_INVALID_ARGUMENT	Port not valid.
SSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortEventOutputWrite()

ssp_err_t R_IOPORT_PortEventOutputWrite (ioport_port_t *port*, ioport_size_t *event_data*, ioport_size_t *mask_value*)

This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite.

Using the event system enables a port state to be stored by this function in advance of being output on the port. The output to the port will occur when the ELC event occurs.

The input value will be written to the specified port when an ELC event configured for that port occurs. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Return values

SSP_SUCCESS	Port event data written.
SSP_ERR_INVALID_ARGUMENT	Port and/or mask not valid. •

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortRead()

ssp_err_t R_IOPORT_PortRead (ioport_port_t *port*, ioport_size_t * *p_port_value*)

Reads the value on an IO port. Implements ioport_api_t::portRead.

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. *

Return values

SSP_SUCCESS	Port read.
SSP_ERR_INVALID_ARGUMENT	Port not valid.
SSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortWrite()

ssp_err_t R_IOPORT_PortWrite (ioport_port_t *port*, ioport_size_t *value*, ioport_size_t *mask*)

Writes to multiple pins on a port. Implements ioport_api_t::portWrite.

The input value will be written to the specified port. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Only the bits with the corresponding bit in the mask value set will be updated. e.g. value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

Return values

SSP_SUCCESS	Port written to.
SSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid.

Note

This function is re-entrant for different ports. This function makes use of the PCNTR3 register to atomically modify the levels on the specified pins on a port.

High bits

Low bits

◆ R_IOPORT_VersionGet()

ssp_err_t R_IOPORT_VersionGet (ssp_version_t * *p_data*)
Returns IOPort HAL driver version. Implements ioport_api_t::versionGet.

Return values

SSP_SUCCESS	Version information read.
SSP_ERR_ASSERTION	The parameter p_data is NULL.

Note

This function is reentrant.

5.1.5.26 IWDT

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Independent Watchdog Timer (IWDT). More...

Data Structures

struct	iwdt_instance_ctrl_t

Functions

ssp_err_t	R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg) Register the IWDT NMI callback. More...
ssp_err_t	R_IWDT_CfgGet (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t *const p_cfg) Read the configuration of the IWDT. Implements wdt_api_t::cfgGet. More...
ssp_err_t	R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl) Refresh the Independent Watchdog Timer. Implements wdt_api_t::refresh. More...
ssp_err_t	R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status) Read the IWDT status flags. More...

ssp_err_t	R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status) Clear the IWDT status and error flags. Implements wdt_api_t::statusClear. More...
ssp_err_t	R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count) Read the current count value of the IWDT. Implements wdt_api_t::counterGet. More...
ssp_err_t	R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout) Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet. More...
ssp_err_t	R_IWDT_VersionGet (ssp_version_t *const p_data) Return IWDT HAL driver version. Implements wdt_api_t::versionGet. More...

Detailed Description

Driver for the Independent Watchdog Timer (IWDT).

Summary

This module supports the Independent Watchdog Timer (IWDT). It implements the WDT Interface. Extends WDT_API HAL layer drivers for interfacing with the Independent Watchdog Timer (IWDT) peripheral.

The IWDT HAL APIs provide the ability to refresh the independent watchdog, read the timer value and read and clear status flags. When used in NMI output mode the callback to be called by the NMI ISR can be registered.

Function Documentation

◆ R_IWDT_CfgGet()

ssp_err_t R_IWDT_CfgGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t *const *p_cfg*)

Read the configuration of the IWDT. Implements wdt_api_t::cfgGet.

Return values

SSP_SUCCESS	IWDT configuration successfully read.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.

Note

This function is reentrant.

Get timeout value from OFS0 register.

◆ R_IWDT_CounterGet()

ssp_err_t R_IWDT_CounterGet (wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count*)

Read the current count value of the IWDT. Implements wdt_api_t::counterGet.

Return values

SSP_SUCCESS	IWDT current count successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

Note

This function is reentrant.

◆ R_IWDT_Open()

ssp_err_t R_IWDT_Open (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg*)

Register the IWDT NMI callback.

Return values

SSP_SUCCESS	IWDT NMI callback successfully configured.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_MODE	An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode.
SSP_ERR_HW_LOCKED	IWDT module has already been called.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Note

This function is not reentrant.

g_iwdt_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures these data structures are not optimized away.

Eliminate toolchain warning when NMI output is not being used.

Lock the IWDT Hardware Resource

Initialize global pointer to WDT for NMI callback use.

Check for NMI output mode

NMI output mode

Enable the IWDT underflow/refresh error interrupt (will generate an NMI).

◆ R_IWDT_Refresh()

ssp_err_t R_IWDT_Refresh (wdt_ctrl_t *const *p_api_ctrl*)

Refresh the Independent Watchdog Timer. Implements wdt_api_t::refresh.

Return values

SSP_SUCCESS	IWDT successfully refreshed.
SSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

Note

This function is reentrant. This function only returns SSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.

◆ R_IWDT_StatusClear()

ssp_err_t R_IWDT_StatusClear (wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status*)

Clear the IWDT status and error flags. Implements wdt_api_t::statusClear.

Return values

SSP_SUCCESS	IWDT flag(s) successfully cleared.
SSP_ERR_ASSERTION	Null pointer as a parameter.
SSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

Note

This function is reentrant.

Write zero to clear flags

◆ R_IWDT_StatusGet()

ssp_err_t R_IWDT_StatusGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status*)

Read the IWDT status flags.

Indicates both status and error conditions.

Return values

SSP_SUCCESS	IWDT status successfully read.
SSP_ERR_ASSERTION	Null pointer as a parameter.
SSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

Note

This function is reentrant. When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags can be read after reset to establish if the IWDT caused the reset. Reading the status and error flags in NMI output mode indicates whether the IWDT generated the NMI interrupt.

◆ R_IWDT_TimeoutGet()

ssp_err_t R_IWDT_TimeoutGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

SSP_SUCCESS	WDT successfully refreshed.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

Note

This function is reentrant. This function must not be called before calling R_WDT_Open().

◆ R_IWDT_VersionGet()

ssp_err_t R_IWDT_VersionGet (ssp_version_t *const *p_data*)

Return IWDT HAL driver version. Implements wdt_api_t::versionGet.

Return values

SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.

Note

This function is reentrant.

iwdt_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » IWDT

#include <r_iwdt.h>

Data Fields

uint32_t	iwdt_open
void const *	p_context
R_IWDT_Type *	p_reg Pointer to register base address.

void(*	p_callback)(wdt_callback_args_t *p_args) Callback provided when a WDT NMI ISR occurs.

Detailed Description

WDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called.

Field Documentation

◆ iwdt_open

uint32_t iwdt_instance_ctrl_t::iwdt_open
Indicates whether the open() API has been successfully called.

◆ p_context

void const* iwdt_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

The documentation for this struct was generated from the following file:

- r_iwdt.h

5.1.5.27 JPEG CODEC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the JPEG CODEC. More...

Data Structures

struct	jpeg_decode_instance_ctrl_t

Functions

ssp_err_t	R_JPEG_Decode_Open (jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_cfg_t const *const p_cfg) Initialize the JPEG Codec module. This function configures the JPEG Codec for decoding operation, sets up the registers for data format and pixel format based on user-supplied configuration parameters.

	Interrupts are enabled to support image size read operation and callback functions. More...
ssp_err_t	R_JPEG_Decode_OutputBufferSet (jpeg_decode_ctrl_t *p_api_ctrl, void *p_output_buffer, uint32_t output_buffer_size) Assign output buffer to the JPEG Codec for storing output data. More...
ssp_err_t	R_JPEG_Decode_LinesDecodedGet (jpeg_decode_ctrl_t *p_api_ctrl, uint32_t *p_lines) Returns the number of lines decoded into the output buffer. More...
ssp_err_t	R_JPEG_Decode_InputBufferSet (jpeg_decode_ctrl_t *const p_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size) Assign input data buffer to JPEG codec for processing. More...
ssp_err_t	R_JPEG_Decode_ImageSubsampleSet (jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) Configure horizontal and vertical subsample. More...
ssp_err_t	R_JPEG_Decode_HorizontalStrideSet (jpeg_decode_ctrl_t *p_api_ctrl, uint32_t horizontal_stride) Configure horizontal stride setting. More...
ssp_err_t	R_JPEG_Decode_Close (jpeg_decode_ctrl_t *p_api_ctrl) Cancel an outstanding JPEG codec operation and close the device. More...
ssp_err_t	R_JPEG_Decode_ImageSizeGet (jpeg_decode_ctrl_t *p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) Obtain the size of the image. This operation is valid during JPEG decoding operation. More...
ssp_err_t	R_JPEG_Decode_StatusGet (jpeg_decode_ctrl_t *p_api_ctrl, jpeg_decode_status_t *p_status) Get the status of the JPEG codec. This function can also be used to poll the device. More...

ssp_err_t	R_JPEG_Decode_PixelFormatGet (jpeg_decode_ctrl_t *p_api_ctrl, jpeg_decode_color_space_t *p_color_space) Get the input pixel format. More...
ssp_err_t	R_JPEG_Decode_VersionGet (ssp_version_t *p_version) Get version of the display interface and GLCD HAL code. More...

Detailed Description

Driver for the JPEG CODEC.

Function Documentation

◆ R_JPEG_Decode_Close()

ssp_err_t R_JPEG_Decode_Close (jpeg_decode_ctrl_t * *p_api_ctrl*)

Cancel an outstanding JPEG codec operation and close the device.

Return values

SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG not opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::eventInfoGet

Clear JPEG JINTE0 interrupt and JINTE1 interrupt.

Disable JEDI and JDTI at NVIC

Power off the JPEG codec.

Reset the jpeg status flag in the driver.

Unlock module at BSP level.

◆ R_JPEG_Decode_HorizontalStrideSet()

ssp_err_t R_JPEG_Decode_HorizontalStrideSet (jpeg_decode_ctrl_t * *p_api_ctrl*, uint32_t *horizontal_stride*)

Configure horizontal stride setting.

Note

Use when the horizontal stride needs to match the image width and the image size is unknown when opening the JPEG driver. (If the image size is known prior to the open call, pass the horizontal stride value in the jpef_cfg_t structure.) After the image size becomes available, use this function to update the horizontal stride value. If the driver must decode one line at a time, the horizontal stride can be set to zero.

Return values

SSP_SUCCESS	Horizontal Stride value is properly configured.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is zero or not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.

Record the horizontal stride value in the control block

Set the horizontal stride.

If the parameters all are set, resume the core to decode.

For the given buffer size, compute number of lines to decode.

◆ R_JPEG_Decode_ImageSizeGet()

ssp_err_t R_JPEG_Decode_ImageSizeGet (jpeg_decode_ctrl_t * *p_api_ctrl*, uint16_t * *p_horizontal_size*, uint16_t * *p_vertical_size*)

Obtain the size of the image. This operation is valid during JPEG decoding operation.

Return values

SSP_SUCCESS	The image size is available and the horizontal and vertical values are stored in the memory pointed to by p_horizontal_size and p_vertical_size.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_IMAGE_SIZE_UNKNOWN	The image size is unknown. More input data may be needed.
SSP_ERR_INVALID_MODE	JPEG Codec module is not decoding.
SSP_ERR_NOT_OPEN	JPEG is not opened.

◆ R_JPEG_Decode_ImageSubsampleSet()

ssp_err_t R_JPEG_Decode_ImageSubsampleSet (jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_subsample_t *horizontal_subsample*, jpeg_decode_subsample_t *vertical_subsample*)

Configure horizontal and vertical subsample.

Note

Use for scaling the decoded image.

Return values

SSP_SUCCESS	Horizontal Stride value is properly configured.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_ARGUMENT	Sub-sample setting is invalid.
SSP_ERR_NOT_OPEN	JPEG not opened.

Update horizontal sub-sample setting.

◆ R_JPEG_Decode_InputBufferSet()

ssp_err_t R_JPEG_Decode_InputBufferSet (jpeg_decode_ctrl_t *const *p_api_ctrl*, void * *p_data_buffer*, uint32_t *data_buffer_size*)

Assign input data buffer to JPEG codec for processing.

Note

After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so JPEG decoding can resume.

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least one line of decoded image data.

Return values

SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the input_buffer is NULL, or the input_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.

Configure the input buffer address.

If the system is idle, start the JPEG engine. This allows the system to obtain image information (image size and input pixel format). This information is needed to drive the decode process later on.

Based on buffer size, detect the in count mode setting. The driver is able to read input data in chunks. However the size of each chunk is limited to BUFFER_MAX_SIZE. Therefore, if the input data size is larger than BUFFER_MAX_SIZE, the driver assumes the entire input data is present, and can be decoded without additional input data. Otherwise, the driver enables input stream feature. This works even if the entire input size is smaller than BUFFER_MAX_SIZE.

◆ R_JPEG_Decode_LinesDecodedGet()

ssp_err_t R_JPEG_Decode_LinesDecodedGet (jpeg_decode_ctrl_t * *p_api_ctrl*, uint32_t * *p_lines*)

Returns the number of lines decoded into the output buffer.

Note

Use this function to retrieve number of image lines written to the output buffer after JPEG decoded a partial image. Combined with the horizontal stride settings and the output pixel format, the application can compute the amount of data to read from the output buffer.

Return values

SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_NOT_OPEN	JPEG not opened.

◆ R_JPEG_Decode_Open()

ssp_err_t R_JPEG_Decode_Open (jpeg_decode_ctrl_t *const *p_api_ctrl*, jpeg_decode_cfg_t const *const *p_cfg*)

Initialize the JPEG Codec module. This function configures the JPEG Codec for decoding operation, sets up the registers for data format and pixel format based on user-supplied configuration parameters. Interrupts are enabled to support image size read operation and callback functions.

Return values

SSP_SUCCESS	JPEG Codec module is properly configured and is ready to take input data.
SSP_ERR_IN_USE	JPEG Codec is already in use.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Verify JPEG Codec is not already used.

Update the common control parameter with the control and JEDI and JDTI callback handler for JPEG decode, the handlers will be called from r_jpeg_common, which implements JPEG JDTI and JEDI ISR for r_jpeg_decode and r_jpeg_encode driver.

Record the configuration settings.

Initialize horizontal stride value.

Initialize output buffer size.

Initialize total_lines_decoded

Initialize horizontal sub-sample setting.

Provide power to the JPEG module.

Clear the image horizontal and vertical size, before starting the JPEG decode

Perform bus reset

Reset the destination buffer address.

Reset the source buffer address.

Reset the horizontal stride.

Configure the JPEG module for decode operation.

Set image format for the decoded image.

If the output pixel format is ARGB8888, also configure the alpha value.

Set the alpha value for the decoded image.

Set the output data format.

The following interrupts are enabled: Interrupt on all errors Interrupt on Image Size

Record user supplied callback routine.

Set the driver status.

All done. Return success.

◆ R_JPEG_Decode_OutputBufferSet()

ssp_err_t R_JPEG_Decode_OutputBufferSet (jpeg_decode_ctrl_t * *p_api_ctrl*, void * *p_output_buffer*, uint32_t *output_buffer_size*)

Assign output buffer to the JPEG Codec for storing output data.

Note

The number of image lines to be decoded depends on the size of the buffer and the horizontal stride settings. Once the output buffer size is known, the horizontal stride value is known, and the input pixel format is known (the input pixel format is obtained by the JPEG decoder from the JPEG headers), the driver automatically computes the number of lines that can be decoded into the output buffer. After these lines are decoded, the JPEG engine pauses and a callback function is triggered, so the application is able to provide the next buffer for the JPEG module to resume the operation.

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least eight lines of decoded image data.

Return values

SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	Invalid buffer size

Set the decoding destination address.

Record the size of the output buffer.

If the image size is not ready yet, the driver does not know the input pixel format. Without that information, the driver is unable to compute the number of lines of image to decode. In this case, the driver would record the output buffer size. Once all the information is ready, the driver would attempt to start the decoding process.

For a given buffer size, compute number of lines to decode if the image size acquisition is known.

If the driver status is IMAGE_SIZE_READY with no other flags, that means the driver just received IMAGE_SIZE. It has not started the decoding process yet.

If Input buffer is set, output buffer is set, and horizontal stride is set, the driver is able to determine the number of lines to decode, and start the decoding operation.

If the current status is OUTPUT_PAUSE, the driver needs to resume the operation.

◆ R_JPEG_Decode_PixelFormatGet()

ssp_err_t R_JPEG_Decode_PixelFormatGet (jpeg_decode_ctrl_t * *p_api_ctrl*, jpeg_decode_color_space_t * *p_color_space*)

Get the input pixel format.

Return values

SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG is not opened.

HW does not report error. Return internal status information.

◆ R_JPEG_Decode_StatusGet()

ssp_err_t R_JPEG_Decode_StatusGet (jpeg_decode_ctrl_t * *p_api_ctrl*, jpeg_decode_status_t * *p_status*)

Get the status of the JPEG codec. This function can also be used to poll the device.

Return values

SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG is not opened.

HW does not report error. Return internal status information.

◆ R_JPEG_Decode_VersionGet()

ssp_err_t R_JPEG_Decode_VersionGet (ssp_version_t * *p_version*)

Get version of the display interface and GLCD HAL code.

Return values

SSP_SUCCESS	Version number
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Note

This function is reentrant.

jpeg_decode_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » JPEG CODEC

#include <r_jpeg_decode.h>

Data Fields

jpeg_decode_status_t	status JPEG Codec module status.
ssp_err_t	error_code JPEG Codec error code (if any).
void(*	p_callback)(jpeg_decode_callback_args_t *p_args) User-supplied callback functions.
void const *	p_extend JPEG Codec hardware dependent configuration */.
void const *	p_context Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.
R_JPEG_Type *	p_reg Pointer to register base address.
jpeg_decode_pixel_format_t	pixel_format Pixel format.
uint32_t	horizontal_stride Horizontal Stride settings.
uint32_t	outbuffer_size out buffer size

uint16_t	total_lines_decoded Track the number of lines decoded so far.
jpeg_decode_subsample_t	horizontal_subsample Horizontal sub-sample setting.

Detailed Description

JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called.

The documentation for this struct was generated from the following file:

- r_jpeg_decode.h

5.1.5.28 JPEG ENCODE

Renesas Synergy Software Package Reference » HAL Layer

Driver for the JPEG CODEC. More...

Data Structures

struct	jpeg_encode_instance_ctrl_t

Macros

#define	JPEG_ENCODE_CODE_VERSION_MAJOR (2U)

Functions

ssp_err_t	R_JPEG_Encode_Open (jpeg_encode_ctrl_t *const p_api_ctrl, jpeg_encode_cfg_t const *const p_cfg) Initialize the JPEG Codec module. This function configures the JPEG Codec for encoding operation, sets up the registers for data format, pixel format, vertical and horizontal resolution stride based on user-supplied configuration parameters. More...
ssp_err_t	R_JPEG_Encode_OutputBufferSet (jpeg_encode_ctrl_t *p_api_ctrl, void *p_output_buffer) Assign output buffer to the JPEG Codec for storing output data.

	More...
ssp_err_t	R_JPEG_Encode_InputBufferSet (jpeg_encode_ctrl_t *const p_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size) Assign input data buffer to JPEG codec for processing. More...
ssp_err_t	R_JPEG_Encode_ImageParameterSet (jpeg_encode_ctrl_t *const p_api_ctrl, jpeg_encode_raw_image_parameters *p_image_parameters) Setup the image parameters to JPEG Codec device. More...
ssp_err_t	R_JPEG_Encode_StatusGet (jpeg_encode_ctrl_t *p_api_ctrl, volatile jpeg_encode_status_t *p_status) Get the status of the JPEG codec. This function can also be used to poll the device. More...
ssp_err_t	R_JPEG_Encode_Close (jpeg_encode_ctrl_t *p_api_ctrl) Cancel an outstanding JPEG codec operation and close the device. More...
ssp_err_t	R_JPEG_Encode_VersionGet (ssp_version_t *p_version) Get version of the display interface and GLCD HAL code. More...

Detailed Description

Driver for the JPEG CODEC.

Macro Definition Documentation

◆ JPEG_ENCODE_CODE_VERSION_MAJOR

#define JPEG_ENCODE_CODE_VERSION_MAJOR (2U)
Configuration for this module

Function Documentation

◆ R_JPEG_Encode_Close()

ssp_err_t R_JPEG_Encode_Close (jpeg_encode_ctrl_t * *p_api_ctrl*)

Cancel an outstanding JPEG codec operation and close the device.

Return values

SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG not opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::eventInfoGet

Clear JPEG JINTE0 interrupt and JINTE1 interrupt.

Disable JEDI and JDTI at NVIC

Power off the JPEG codec.

Reset the jpeg status flag in the driver.

Unlock module at BSP level.

◆ R_JPEG_Encode_ImageParameterSet()

ssp_err_t R_JPEG_Encode_ImageParameterSet (jpeg_encode_ctrl_t *const *p_api_ctrl*, jpeg_encode_raw_image_parameters * *p_image_parameters*)

Setup the image parameters to JPEG Codec device.

Note

Image parameters needs to be set before the setting the input buffer.DO NOT call this function during the JPEG Codec operation.

Return values

SSP_SUCCESS	Image parameter is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL,
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.
SSP_ERR_INVALID_ARGUMENT	Horizontal and Vertical resolution is invalid or zero.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made.

Do not change the JPEG Codec image setting while JPEG Codec is in progress

Record the horizontal stride and vertical size value in the control block

Set the horizontal stride.

Set the image horizontal and vertical size.

◆ R_JPEG_Encode_InputBufferSet()

ssp_err_t R_JPEG_Encode_InputBufferSet (jpeg_encode_ctrl_t *const *p_api_ctrl*, void * *p_data_buffer*, uint32_t *data_buffer_size*)

Assign input data buffer to JPEG codec for processing.

Note

1.After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so JPEG encoding can resume. 2.Image size should be greater or equal to minimum coded unit (MCU) for YCbCr422 (only supported color space) the MCU is 8 lines by 16 pixel (where 1 pixel = 2 bytes) hence size can not be less than 8x16x2 = 256

The JPEG encoding operation automatically starts after setting the input buffer.

Return values

SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the

	pointer to the p_data_buffer is NULL.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address or image line to encode or size of buffer is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made, set output buffer first
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG Codec

Validate the the size : JPEG Codec can process minimum up to 8 lines by 16 pixel for YCbCr422 meaning 8x16x2 = 256

Calculate the number of lines to be encode

JPEG Codec required Lines to be byte aligned

Check, If output image buffer is set or not

Configure the input buffer address.

if JPEG is just opened or completed one image, make DONE and IDLE status flag zero to encode next image

Remove the Done and IDLE status flag

Set the driver status to JPEG_ENCODE_STATUS_RUNNING.

Start the encoder

JPEG is PAUSE for next chunk of image

Clear internal status information.

Set RUNNING status

Resume the count mode

JPEG is running Notify the user, return error

◆ R_JPEG_Encode_Open()

ssp_err_t R_JPEG_Encode_Open (jpeg_encode_ctrl_t *const *p_api_ctrl*, jpeg_encode_cfg_t const *const *p_cfg*)

Initialize the JPEG Codec module. This function configures the JPEG Codec for encoding operation, sets up the registers for data format, pixel format, vertical and horizontal resolution stride based on user-supplied configuration parameters.

Return values

SSP_SUCCESS	JPEG Codec module is properly configured and is ready to take input data.
SSP_ERR_IN_USE	JPEG Codec is already in use.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Verify JPEG Codec is not already used.

Update the common control parameter with the control and JDEI and JDTI callback handler for JPEG encode, the handlers will be called from r_jpeg_common, which implements JPEG JDTI and JDEI ISR for r_jpeg_decode and r_jpeg_encode driver.

Get the JDTI event information from FMI

Get the vector table information of JDTI event

Record the JPEG Encoder ctrl and internal ISR callback handler to JDTI event vector table

Get the JEDI event information from FMI

Get the vector table information of JEDI event

Record the JPEG Encoder ctrl and internal ISR callback handler to JEDI event vector table

Provide power to the JPEG module.

Perform bus reset

Reset the destination buffer address.

Reset the source buffer address.

Set the horizontal stride.

Set the image horizontal and vertical size.

Configure the JPEG module for encode operation.

Set the output data format.

Set input pixel format for JPEG Encoder NOTE : only ycbcr422 is valid for encoder

Quantization table setting NOTE: Table 0(zero) is used for Luminance and Table 1(one) is used for Chrominance - Cr and Cb component

Upload Luminance and Chrominance table to JPEG Codec

Huffman table setting

Upload the Huffman table to JPEG Codec

Reset Marker setting

Record image parameters to ctrl

Set the driver status.

Record the user context information

Record user supplied callback routine.

Enabled JPEG Compression data transfer complete interrupt and Count mode Interrupt

Enable the JDTI and JDEI interrupts

All done. Return success.

◆ R_JPEG_Encode_OutputBufferSet()

ssp_err_t R_JPEG_Encode_OutputBufferSet (jpeg_encode_ctrl_t * *p_api_ctrl*, void * *p_output_buffer*)

Assign output buffer to the JPEG Codec for storing output data.

Note

Buffer size should be sufficient to hold the encoded jpeg image.

Return values

SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to changed during codec operation

Output buffer cannot be change during codec operation

Set the encoding destination address.

◆ R_JPEG_Encode_StatusGet()

ssp_err_t R_JPEG_Encode_StatusGet (jpeg_encode_ctrl_t * *p_api_ctrl*, volatile jpeg_encode_status_t * *p_status*)

Get the status of the JPEG codec. This function can also be used to poll the device.

Return values

SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

HW does not report error. Return internal status information.

◆ R_JPEG_Encode_VersionGet()

ssp_err_t R_JPEG_Encode_VersionGet (ssp_version_t * *p_version*)

Get version of the display interface and GLCD HAL code.

Return values

SSP_SUCCESS	Version number
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Note

This function is reentrant.

jpeg_encode_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » JPEG ENCODE

#include <r_jpeg_encode.h>

Data Fields

volatile jpeg_encode_status_t	status JPEG Codec module status.
void(*	p_callback)(jpeg_encode_callback_args_t *p_args) User-supplied callback functions.

void const *	p_extend JPEG Codec hardware dependent configuration */.
void const *	p_context Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.
R_JPEG_Type *	p_reg Pointer to register base address.
uint32_t	horizontal_stride Horizontal Stride settings (Line offset).
uint32_t	output_buffer_size out buffer size
uint16_t	lines_to_encoded Number of lines to encode.
uint16_t	vertical_resolution vertical size
uint16_t	encoded_lines Number of lines encoded.

Detailed Description

JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called.

The documentation for this struct was generated from the following file:

- r_jpeg_encode.h

5.1.5.29 Key Interrupts

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Key Interrupt Function. More...

Data Structures

struct	kint_instance_ctrl_t

Functions

ssp_err_t	R_KINT_KEYMATRIX_Open (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_cfg_t const *const p_cfg) Power on KINT, handle required initialization described in hardware manual. Implements keymatrix_api_t::open. More...
ssp_err_t	R_KINT_KEYMATRIX_Close (keymatrix_ctrl_t *const p_api_ctrl) Disable KINT. Implements keymatrix_api_t::close. More...
ssp_err_t	R_KINT_KEYMATRIX_Enable (keymatrix_ctrl_t *const p_api_ctrl) Enable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements keymatrix_api_t::enable. More...
ssp_err_t	R_KINT_KEYMATRIX_Disable (keymatrix_ctrl_t *const p_api_ctrl) Disable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements keymatrix_api_t::disable. More...
ssp_err_t	R_KINT_KEYMATRIX_TriggerSet (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_trigger_t hw_trigger) Set trigger edge (falling or rising) provided. Implements keymatrix_api_t::triggerSet. More...
ssp_err_t	R_KINT_VersionGet (ssp_version_t *const p_version) Set driver version based on compile time macros. More...

Detailed Description

Driver for the Key Interrupt Function.

The Key input driver can be used for one to eight channels or in a matrix format. This module implements the following interface: Key Matrix Interface.

Function Documentation

◆ R_KINT_KEYMATRIX_Close()

ssp_err_t R_KINT_KEYMATRIX_Close (keymatrix_ctrl_t *const *p_api_ctrl*)

Disable KINT. Implements keymatrix_api_t::close.

End of function R_KINT_KEYMATRIX_Open The Close function disables the interrupts in the peripheral and the NVIC and clears any pending interrupt requests. It also releases the hardware lock on the API.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

Disable interrupt in ICU

Disable interrupts in the KINT peripheral

Clear any pending interrupt requests in the KINT peripheral

Clear the Interrupt Request bit

Clear stored internal driver data

Release the lock

◆ R_KINT_KEYMATRIX_Disable()

ssp_err_t R_KINT_KEYMATRIX_Disable (keymatrix_ctrl_t *const *p_api_ctrl*)

Disable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements keymatrix_api_t::disable.

This function disables interrupts for the KINT peripheral both at the interrupt level and in the NVIC. All the channels specified by R_KINT_KEYMATRIX_Open are disabled.

Return values

SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Disable interrupts in the KINT peripheral

Clear any pending interrupt requests in the KINT peripheral

Disable interrupt in the ICU

◆ R_KINT_KEYMATRIX_Enable()

ssp_err_t R_KINT_KEYMATRIX_Enable (keymatrix_ctrl_t *const *p_api_ctrl*)

Enable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements keymatrix_api_t::enable.

This function enables interrupts for the KINT peripheral both at the interrupt level and in the NVIC after clearing any pending requests in the KINT and ICU peripheral. All the channels specified by R_KINT_KEYMATRIX_Open are enabled.

Return values

SSP_SUCCESS	Interrupt enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The peripheral is not opened.

Clear any pending interrupt requests in the KINT peripheral

Clear the Interrupt Request Flag in the ICU

Enable interrupt for the selected channels after casting since KRM is an 8 bit register

Enable interrupt

◆ R_KINT_KEYMATRIX_Open()

ssp_err_t R_KINT_KEYMATRIX_Open (keymatrix_ctrl_t *const *p_api_ctrl*, keymatrix_cfg_t const *const *p_cfg*)

Power on KINT, handle required initialization described in hardware manual. Implements keymatrix_api_t::open.

The Open function configures all the Key Input (KINT) channels and provides a handle for use with the rest of the KINT API functions. This function must be called at least once prior to calling any other KINT API functions. After the peripheral is initialized, the Open function should not be called again without first calling the Close function.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_cfg, or p_ctrl or the callback.
SSP_ERR_INVALID_ARGUMENT	One of the following may be invalid: • p_cfg->channel: must be between 0 and KINT_MAX_NUM_CHANNELS • p_cfg->trigger not a valid value.
SSP_ERR_HW_LOCKED	The API has already been opened. It must be closed before it can be opened again.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is not reentrant.

Check to see that the arguments passed are not null pointers

Grab the hardware lock. If successful this indicates that the open was not previously called.

Disable interrupts in the KINT peripheral

Clear any pending interrupt requests in the KINT peripheral

Clear the Interrupt Request in the ICU

Configure the trigger edge. The trigger edge can be modified in the TriggerSet function later if desired

KEYMATRIX_TRIG_RISING condition

Configure the usage of key interrupt flags

Store the callback and context information

If interrupts are to be enabled now, set it up for the selected channels. The channels can be changed later in the enable function but to modify the callback and context, the API has to be closed and reopened with the new callback and context.

Note

The KINT hardware only supports a single interrupt for all channels

Enable interrupt for the selected channels after casting since KRM is an 8 bit register

Enable interrupt

Store number of channels for use to the control block to use it later

Mark driver as open by initializing it to "KINT" in its ASCII equivalent.

◆ R_KINT_KEYMATRIX_TriggerSet()

ssp_err_t R_KINT_KEYMATRIX_TriggerSet (keymatrix_ctrl_t *const *p_api_ctrl*, keymatrix_trigger_t *hw_trigger*)

Set trigger edge (falling or rising) provided. Implements keymatrix_api_t::triggerSet.

This function changes trigger sense at run-time. The change is applied to all the channels specified by R_KINT_KEYMATRIX_Open.

Return values

SSP_SUCCESS	Trigger value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARGUMENT	Trigger input invalid. hw_trigger must be one of the options from button_trigger_t.
SSP_ERR_NOT_OPEN	The channel is not opened.

Note

This function expects to be called when the driver is disabled (the driver state before R_KINT_KEYMATRIX_Enable is called if the driver is opened in the non-auto start mode, or after R_KINT_KEYMATRIX_Disable is called if the driver is opened in the auto start mode). The driver does not restrict to call this API when the driver is enabled but users need to make sure the edge detection sense is instantly changed by this API call.

Configure the trigger edge

KEYMATRIX_TRIG_RISING condition

Configure the usage of key interrupt flags

◆ R_KINT_VersionGet()

ssp_err_t R_KINT_VersionGet (ssp_version_t *const *p_version*)

Set driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

kint_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Key Interrupts

#include <r_kint.h>

Data Fields

R_KINT_Type *	p_reg Pointer to register base address.
keymatrix_channels_t	channels Channel bitmask.
IRQn_Type	irq Interrupt priority number.
uint32_t	open Flag to determine if the device is open.

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when keymatrix_api_t::open is called.

The documentation for this struct was generated from the following file:

- r_kint.h

5.1.5.30 LPMV2 S124

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,

```
LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
  LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
  LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
  LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
  LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
  LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

enum

```
lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

enum

```
lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

enum

```
lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

enum lpmv2_standby_wake_source_t { ,
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U,
LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U
}

Functions

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const p_version)

Get the driver version based on compile time macros. More...

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization. More...

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg)

Configure a low power mode. More...

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro. More...

ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t

Snooze cancel control

Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfully applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S124

Macros

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ LPMV2_CFG_PARAM_CHECKING_ENABLE

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)
Specify whether to include code for API parameter checking. Valid settings include: • BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h • 1 : Includes parameter checking • 0 : Compiles out parameter checking

5.1.5.31 LPMV2 S128

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

```
enum    lpmv2_snooze_request_t { ,
          LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,
        LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
        LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
        LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
          LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
        LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
        LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
        LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
          LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
        LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
        LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
        LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
          LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
        LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
        LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
        LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
          LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
        LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
        LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
        LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
          LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
        LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
```

```
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum  lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum  lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum  lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum  lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
  LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
```

	LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfully applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S128

Macros

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ LPMV2_CFG_PARAM_CHECKING_ENABLE

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)
Specify whether to include code for API parameter checking. Valid settings include: • BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h • 1 : Includes parameter checking • 0 : Compiles out parameter checking

5.1.5.32 LPMV2 S1JA

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

```
enum    lpmv2_snooze_request_t { ,
          LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,
        LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
        LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
        LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
          LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
        LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
        LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
        LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
          LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
        LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
        LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
        LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
          LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
        LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
        LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
        LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
          LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
        LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
        LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
        LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
          LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
        LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
```

```
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum  lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum  lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum  lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum  lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
  LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
```

	LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfully applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S1JA

Macros

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ LPMV2_CFG_PARAM_CHECKING_ENABLE

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)
Specify whether to include code for API parameter checking. Valid settings include: • BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h • 1 : Includes parameter checking • 0 : Compiles out parameter checking

5.1.5.33 LPMV2 S3A1

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

```
enum    lpmv2_snooze_request_t { ,
          LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,
        LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
        LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
        LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
          LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
        LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
        LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
        LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
          LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
        LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
        LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
        LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
          LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
        LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
        LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
        LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
          LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
        LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
        LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
        LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
          LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
        LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
```

```
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum  lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum  lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum  lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum  lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
  LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
```

	LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfully applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S3A1

Macros

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ LPMV2_CFG_PARAM_CHECKING_ENABLE

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)
Specify whether to include code for API parameter checking. Valid settings include: • BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h • 1 : Includes parameter checking • 0 : Compiles out parameter checking

5.1.5.34 LPMV2 S3A3

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

```
enum    lpmv2_snooze_request_t { ,
          LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,
        LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
        LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
        LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
          LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
        LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
        LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
        LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
          LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
        LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
        LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
        LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
          LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
        LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
        LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
        LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
          LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
        LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
        LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
        LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
          LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
        LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
        LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
```

```
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum  lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum  lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum  lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum  lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
  LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
```

	LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S3A3

Macros

#define	LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ **LPMV2_CFG_PARAM_CHECKING_ENABLE**

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Specify whether to include code for API parameter checking. Valid settings include:

- BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h
- 1 : Includes parameter checking
- 0 : Compiles out parameter checking

5.1.5.35 LPMV2 S3A6

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U , LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U, LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U, LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U, LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U, LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U, LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U, LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U, LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U, LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U, LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U, LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,

```
  LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
  LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
  LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
```

LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U,
LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U
}

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

5.1.5.36 LPMV2 S3A7

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U , LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U, LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U, LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U, LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U, LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U, LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U, LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U, LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U, LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U, LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U, LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U, LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U, LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U, LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U, LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U, LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U, LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U, LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U, LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U, LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U, LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U, LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U, LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U, LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U }
enum	lpmv2_snooze_end_t { , LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U , LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U, LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U, LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U, LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U, LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,

	LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U, LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U, LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U, LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U }
enum	lpmv2_snooze_cancel_t { , LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79), LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80), LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85), LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86), LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376), LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377), LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41), LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134), LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18) }
enum	lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U, LPMV2_SNOOZE_DTC_ENABLE = 1U }
enum	lpmv2_standby_wake_source_t { , LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U , LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U , LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U, LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U, LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U, LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U, LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U, LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U, LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U, LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U, LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U, LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U, LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U, LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U, LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U, LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U, LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U, LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U, LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U, LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }

enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S3A7

Macros

#define	LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ **LPMV2_CFG_PARAM_CHECKING_ENABLE**

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Specify whether to include code for API parameter checking. Valid settings include:

- BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h
- 1 : Includes parameter checking
- 0 : Compiles out parameter checking

5.1.5.37 LPMV2 S5D3

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U , LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U, LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U, LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U, LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U, LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U, LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U, LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U, LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U, LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U, LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U, LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,

```
  LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
  LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
  LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U,
```

LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U,
LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U,
LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U,
LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U,
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U,
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U,
LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U,
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U,
LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U
}

enum lpmv2_io_port_t { , LPMV2_IO_PORT_RESET = 0U,
LPMV2_IO_PORT_NO_CHANGE = 1U }

enum lpmv2_power_supply_t { , LPMV2_POWER_SUPPLY_DEEPCUT0 = 0U,
LPMV2_POWER_SUPPLY_DEEPCUT1 = 1U,
LPMV2_POWER_SUPPLY_DEEPCUT3 = 3UL }

enum lpmv2_deep_standby_cancel_edge_t { ,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING =
0x00000001U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING =
0x00000002U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING =
0x00000004U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING =
0x00000008U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING =
0x00000010U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING =
0x00000020U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING =
0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING =

0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING = 0x00000800U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING = 0x00001000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING = 0x00002000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING = 0x00004000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING = 0x00008000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING = 0x00010000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING = 0x00020000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING = 0x00100000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING = 0U
}

enum lpmv2_deep_standby_cancel_source_t { ,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 = 0x00000001U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 = 0x00000002U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 = 0x00000004U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 = 0x00000008U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 = 0x00000010U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 = 0x00000020U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 = 0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 = 0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 = 0x00000800U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 = 0x00001000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 = 0x00002000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 = 0x00004000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15 = 0x00008000U,

	LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1 = 0x00010000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2 = 0x00020000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL = 0x00040000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM = 0x00080000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI = 0x00100000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS = 0x01000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS = 0x02000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1 = 0x04000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_deep_standby_cancel_edge_t

enum lpmv2_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_	IRQ7-DS Pin Rising Edge (S5D3, S5D5, S5D9,

RISING	S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ15-DS Pin Rising Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_	

RISING	LVD1 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_deep_standby_cancel_source_t

enum lpmv2_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11 (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14 (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15 (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_io_port_t

enum lpmv2_io_port_t	
I/O port state after Deep Software Standby mode (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state.
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode.

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_power_supply_t

enum lpmv2_power_supply_t	
Power supply control (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

lpmv2_mcu_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S124Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S128Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S1JARenesas Synergy Software Package Reference » HAL Layer » | LPMV2 S3A1 Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S3A3Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S3A6Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S3A7Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S5D3Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S5D5Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S5D9Renesas Synergy Software Package Reference » HAL Layer » | LPMV2 S7G2

#include <r_lpmv2_s7g2.h>

Data Fields

lpmv2_standby_wake_source_bits_t	standby_wake_sources
lpmv2_snooze_request_t	snooze_request_source
lpmv2_snooze_end_bits_t	snooze_end_sources
lpmv2_snooze_cancel_t	snooze_cancel_sources
lpmv2_snooze_dtc_t	dtc_state_in_snooze
lpmv2_output_port_enable_t	output_port_enable
lpmv2_io_port_t	io_port_state
lpmv2_power_supply_t	power_supply_state
lpmv2_deep_standby_cancel_source_bits_t	deep_standby_cancel_source
lpmv2_deep_standby_cancel_edge_bits_t	deep_standby_cancel_edge

Detailed Description

MCU-specific configuration structure

Field Documentation

◆ deep_standby_cancel_edge

lpmv2_deep_standby_cancel_edge_bits_t lpmv2_mcu_cfg_t::deep_standby_cancel_edge
Signal edges for the sources that can trigger exit from deep standby (S5D3, S5D5, S5D9, S7G2).

◆ deep_standby_cancel_source

lpmv2_deep_standby_cancel_source_bits_t lpmv2_mcu_cfg_t::deep_standby_cancel_source
Sources that can trigger exit from deep standby (S5D3, S5D5, S5D9, S7G2).

◆ dtc_state_in_snooze

lpmv2_snooze_dtc_t lpmv2_mcu_cfg_t::dtc_state_in_snooze
State of DTC in snooze mode, enabled or disabled (all Synergy MCUs).

◆ io_port_state

lpmv2_io_port_t lpmv2_mcu_cfg_t::io_port_state
IO port state in deep standby (maintained or reset) (S5D3, S5D5, S5D9, S7G2).

◆ output_port_enable

lpmv2_output_port_enable_t lpmv2_mcu_cfg_t::output_port_enable
Output port enabled/disabled in standby and deep standby (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).

◆ power_supply_state

lpmv2_power_supply_t lpmv2_mcu_cfg_t::power_supply_state
Internal power supply state in standby and deep standby (deepcut) (S5D3, S5D5, S5D9, S7G2).

◆ snooze_cancel_sources

lpmv2_snooze_cancel_t lpmv2_mcu_cfg_t::snooze_cancel_sources
list of snooze cancel sources (all Synergy MCUs).

◆ snooze_end_sources

lpmv2_snooze_end_bits_t lpmv2_mcu_cfg_t::snooze_end_sources
Bitwise list of snooze end sources (all Synergy MCUs).

◆ snooze_request_source

lpmv2_snooze_request_t lpmv2_mcu_cfg_t::snooze_request_source
Snooze request source (all Synergy MCUs).

◆ standby_wake_sources

lpmv2_standby_wake_source_bits_t lpmv2_mcu_cfg_t::standby_wake_sources
Bitwise list of sources to wake from standby (all Synergy MCUs).

The documentation for this struct was generated from the following files:

- r_lpmv2_s124.h
- r_lpmv2_s128.h
- r_lpmv2_s1ja.h
- r_lpmv2_s3a1.h
- r_lpmv2_s3a3.h
- r_lpmv2_s3a6.h
- r_lpmv2_s3a7.h
- r_lpmv2_s5d3.h
- r_lpmv2_s5d5.h
- r_lpmv2_s5d9.h
- r_lpmv2_s7g2.h

5.1.5.38 LPMV2 S5D5

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

```
enum lpmv2_snooze_request_t { ,
  LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U ,
LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U,
LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U,
LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U,
  LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U,
LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U,
LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U,
LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,
  LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
  LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
  LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
  LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

enum lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U, LPMV2_SNOOZE_DTC_ENABLE = 1U }

enum lpmv2_standby_wake_source_t { , LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U , LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U , LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U, LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U, LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U, LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U, LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U, LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U, LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U, LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U, LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U, LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U, LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U, LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U, LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U, LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U, LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U, LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U, LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U, LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }

enum lpmv2_io_port_t { , LPMV2_IO_PORT_RESET = 0U, LPMV2_IO_PORT_NO_CHANGE = 1U }

enum lpmv2_power_supply_t { , LPMV2_POWER_SUPPLY_DEEPCUT0 = 0U, LPMV2_POWER_SUPPLY_DEEPCUT1 = 1U, LPMV2_POWER_SUPPLY_DEEPCUT3 = 3UL }

enum lpmv2_deep_standby_cancel_edge_t { , LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING = 0x00000001U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING = 0x00000002U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING =

0x00000004U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING =
0x00000008U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING =
0x00000010U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING =
0x00000020U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING =
0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING =
0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING =
0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING =
0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING =
0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING =
0x00000800U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING =
0x00001000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING =
0x00002000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING =
0x00004000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING =
0x00008000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING =
0x00010000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING =
0x00020000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING =
0x00100000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING = 0U
}

enum lpmv2_deep_standby_cancel_source_t { ,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 = 0x00000001U,

	LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 = 0x00000002U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 = 0x00000004U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 = 0x00000008U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 = 0x00000010U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 = 0x00000020U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 = 0x00000040U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 = 0x00000080U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 = 0x00000100U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 = 0x00000200U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 = 0x00000400U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 = 0x00000800U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 = 0x00001000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 = 0x00002000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 = 0x00004000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15 = 0x00008000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1 = 0x00010000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2 = 0x00020000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL = 0x00040000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM = 0x00080000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI = 0x00100000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS = 0x01000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS = 0x02000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1 = 0x04000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_deep_standby_cancel_edge_t

enum lpmv2_deep_standby_cancel_edge_t

Deep Standby Interrupt Edge

Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge (S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ15-DS Pin Rising Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_deep_standby_cancel_source_t

enum lpmv2_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4 (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14 (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15 (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_io_port_t

enum lpmv2_io_port_t	
I/O port state after Deep Software Standby mode (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state.
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode.

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_power_supply_t

enum lpmv2_power_supply_t	
Power supply control (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S5D5

Macros

#define	LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ **LPMV2_CFG_PARAM_CHECKING_ENABLE**

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Specify whether to include code for API parameter checking. Valid settings include:

- BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h
- 1 : Includes parameter checking
- 0 : Compiles out parameter checking

5.1.5.39 LPMV2 S5D9

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U , LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U, LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U, LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U, LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U, LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U, LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U, LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,

	LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U, LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U, LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U, LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U, LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U, LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U, LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U, LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U, LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U, LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U, LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U, LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U, LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U, LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U, LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U, LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U, LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U }
enum	lpmv2_snooze_end_t { , LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U , LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U, LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U, LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U, LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U, LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U, LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U, LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U, LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U, LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U }
enum	lpmv2_snooze_cancel_t { , LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79), LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80), LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85), LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86), LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376), LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377), LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41), LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134), LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18) }
enum	lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U, LPMV2_SNOOZE_DTC_ENABLE = 1U }
enum	lpmv2_standby_wake_source_t { , LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U , LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U , LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U, LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U, LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U, LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,

	LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U, LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U, LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U, LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U, LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U, LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U, LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U, LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U, LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U, LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U, LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U, LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U, LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U, LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }
enum	lpmv2_io_port_t { , LPMV2_IO_PORT_RESET = 0U, LPMV2_IO_PORT_NO_CHANGE = 1U }
enum	lpmv2_power_supply_t { , LPMV2_POWER_SUPPLY_DEEPCUT0 = 0U, LPMV2_POWER_SUPPLY_DEEPCUT1 = 1U, LPMV2_POWER_SUPPLY_DEEPCUT3 = 3UL }
enum	lpmv2_deep_standby_cancel_edge_t { , LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING = 0x00000001U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING = 0x00000002U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING = 0x00000004U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING = 0x00000008U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING = 0x00000010U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING = 0x00000020U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING = 0U,

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING = 0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING = 0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING = 0x00000800U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING = 0x00001000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING = 0x00002000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING = 0x00004000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING = 0x00008000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING = 0x00010000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING = 0x00020000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING = 0x00100000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING = 0U
}

enum lpmv2_deep_standby_cancel_source_t { ,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 = 0x00000001U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 = 0x00000002U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 = 0x00000004U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 = 0x00000008U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 = 0x00000010U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 = 0x00000020U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 = 0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 = 0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 = 0x00000800U,

	LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 = 0x00001000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 = 0x00002000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 = 0x00004000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15 = 0x00008000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1 = 0x00010000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2 = 0x00020000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL = 0x00040000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM = 0x00080000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI = 0x00100000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS = 0x01000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS = 0x02000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1 = 0x04000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_deep_standby_cancel_edge_t

enum lpmv2_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15-DS Pin Rising Edge (S7G2).

_RISING	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_deep_standby_cancel_source_t

enum lpmv2_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9 (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14 (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15 (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_io_port_t

enum lpmv2_io_port_t	
I/O port state after Deep Software Standby mode (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state.
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode.

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_power_supply_t

enum lpmv2_power_supply_t	
Power supply control (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configures MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S5D9

Macros

#define	LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ **LPMV2_CFG_PARAM_CHECKING_ENABLE**

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Specify whether to include code for API parameter checking. Valid settings include:

- BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h
- 1 : Includes parameter checking
- 0 : Compiles out parameter checking

5.1.5.40 LPMV2 S7G2

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Power Modes. More...

Modules

	Build Time Configurations

Data Structures

struct	lpmv2_mcu_cfg_t

Enumerations

enum	lpmv2_snooze_request_t { , LPMV2_SNOOZE_REQUEST_ACMPLP = 0x00800000U , LPMV2_SNOOZE_REQUEST_RXD0_FALLING = 0x00000000U, LPMV2_SNOOZE_REQUEST_IRQ0 = 0x00000001U, LPMV2_SNOOZE_REQUEST_IRQ1 = 0x00000002U, LPMV2_SNOOZE_REQUEST_IRQ2 = 0x00000004U, LPMV2_SNOOZE_REQUEST_IRQ3 = 0x00000008U, LPMV2_SNOOZE_REQUEST_IRQ4 = 0x00000010U, LPMV2_SNOOZE_REQUEST_IRQ5 = 0x00000020U,

```
  LPMV2_SNOOZE_REQUEST_IRQ6 = 0x00000040U,
LPMV2_SNOOZE_REQUEST_IRQ7 = 0x00000080U,
LPMV2_SNOOZE_REQUEST_IRQ8 = 0x00000100U,
LPMV2_SNOOZE_REQUEST_IRQ9 = 0x00000200U,
  LPMV2_SNOOZE_REQUEST_IRQ10 = 0x00000400U,
LPMV2_SNOOZE_REQUEST_IRQ11 = 0x00000800U,
LPMV2_SNOOZE_REQUEST_IRQ12 = 0x00001000U,
LPMV2_SNOOZE_REQUEST_IRQ13 = 0x00002000U,
  LPMV2_SNOOZE_REQUEST_IRQ14 = 0x00004000U,
LPMV2_SNOOZE_REQUEST_IRQ15 = 0x00008000U,
LPMV2_SNOOZE_REQUEST_KEY = 0x00020000U,
LPMV2_SNOOZE_REQUEST_ACMPHS0 = 0x00400000U,
  LPMV2_SNOOZE_REQUEST_RTC_ALARM = 0x01000000U,
LPMV2_SNOOZE_REQUEST_RTC_PERIOD = 0x02000000U,
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW = 0x10000000U,
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A = 0x20000000U,
  LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B = 0x40000000U
}
```

```
enum  lpmv2_snooze_end_t { ,
  LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH = 0x80U ,
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES = 0x00U,
LPMV2_SNOOZE_END_AGT1_UNDERFLOW = 0x01U,
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE = 0x02U,
  LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED = 0x04U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH = 0x08U,
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH = 0x10U,
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH = 0x20U,
  LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH = 0x40U,
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH = 0x80U
}
```

```
enum  lpmv2_snooze_cancel_t { ,
  LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM = (79),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM = (80),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM = (85),
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM = (86),
  LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM = (376),
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI = (377),
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE = (41),
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI = (134),
  LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN = (18)
}
```

```
enum  lpmv2_snooze_dtc_t { , LPMV2_SNOOZE_DTC_DISABLE = 0U,
LPMV2_SNOOZE_DTC_ENABLE = 1U }
```

```
enum  lpmv2_standby_wake_source_t { ,
  LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0 = 0x00800000U ,
LPMV2_STANDBY_WAKE_SOURCE_VBATT = 0x00100000U ,
LPMV2_STANDBY_WAKE_SOURCE_IRQ0 = 0x00000001U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ1 = 0x00000002U,
  LPMV2_STANDBY_WAKE_SOURCE_IRQ2 = 0x00000004U,
LPMV2_STANDBY_WAKE_SOURCE_IRQ3 = 0x00000008U,
```

	LPMV2_STANDBY_WAKE_SOURCE_IRQ4 = 0x00000010U, LPMV2_STANDBY_WAKE_SOURCE_IRQ5 = 0x00000020U, LPMV2_STANDBY_WAKE_SOURCE_IRQ6 = 0x00000040U, LPMV2_STANDBY_WAKE_SOURCE_IRQ7 = 0x00000080U, LPMV2_STANDBY_WAKE_SOURCE_IRQ8 = 0x00000100U, LPMV2_STANDBY_WAKE_SOURCE_IRQ9 = 0x00000200U, LPMV2_STANDBY_WAKE_SOURCE_IRQ10 = 0x00000400U, LPMV2_STANDBY_WAKE_SOURCE_IRQ11 = 0x00000800U, LPMV2_STANDBY_WAKE_SOURCE_IRQ12 = 0x00001000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ13 = 0x00002000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ14 = 0x00004000U, LPMV2_STANDBY_WAKE_SOURCE_IRQ15 = 0x00008000U, LPMV2_STANDBY_WAKE_SOURCE_IWDT = 0x00010000U, LPMV2_STANDBY_WAKE_SOURCE_KEY = 0x00020000U, LPMV2_STANDBY_WAKE_SOURCE_LVD1 = 0x00040000U, LPMV2_STANDBY_WAKE_SOURCE_LVD2 = 0x00080000U, LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0 = 0x00400000U, LPMV2_STANDBY_WAKE_SOURCE_RTCALM = 0x01000000U, LPMV2_STANDBY_WAKE_SOURCE_RTCPRD = 0x02000000U, LPMV2_STANDBY_WAKE_SOURCE_USBHS = 0x04000000U, LPMV2_STANDBY_WAKE_SOURCE_USBFS = 0x08000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1UD = 0x10000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CA = 0x20000000U, LPMV2_STANDBY_WAKE_SOURCE_AGT1CB = 0x40000000U, LPMV2_STANDBY_WAKE_SOURCE_IIC0 = 0x80000000U }
enum	lpmv2_io_port_t { , LPMV2_IO_PORT_RESET = 0U, LPMV2_IO_PORT_NO_CHANGE = 1U }
enum	lpmv2_power_supply_t { , LPMV2_POWER_SUPPLY_DEEPCUT0 = 0U, LPMV2_POWER_SUPPLY_DEEPCUT1 = 1U, LPMV2_POWER_SUPPLY_DEEPCUT3 = 3UL }
enum	lpmv2_deep_standby_cancel_edge_t { , LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING = 0x00000001U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING = 0x00000002U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING = 0x00000004U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING = 0x00000008U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING = 0x00000010U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING = 0U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING = 0x00000020U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING = 0U,

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING = 0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING = 0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING = 0x00000800U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING = 0x00001000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING = 0x00002000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING = 0x00004000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING = 0x00008000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING = 0x00010000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING = 0x00020000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING = 0x00100000U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING = 0U
}

enum lpmv2_deep_standby_cancel_source_t { ,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY = 0U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 = 0x00000001U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 = 0x00000002U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 = 0x00000004U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 = 0x00000008U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 = 0x00000010U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 = 0x00000020U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 = 0x00000040U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 = 0x00000080U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 = 0x00000100U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 = 0x00000200U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 = 0x00000400U,
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 = 0x00000800U,

	LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 = 0x00001000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 = 0x00002000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 = 0x00004000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15 = 0x00008000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1 = 0x00010000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2 = 0x00020000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL = 0x00040000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM = 0x00080000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI = 0x00100000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS = 0x01000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS = 0x02000000U, LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1 = 0x04000000U }
enum	lpmv2_output_port_enable_t { , LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE = 0U, LPMV2_OUTPUT_PORT_ENABLE_RETAIN = 1U }

Functions

ssp_err_t	R_LPMV2_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...
ssp_err_t	R_LPMV2_Init (void) Perform any necessary initialization. More...
ssp_err_t	R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const p_cfg) Configure a low power mode. More...
ssp_err_t	R_LPMV2_LowPowerModeEnter (void) Enter low power mode (sleep/standby/deep standby) using WFI macro. More...
ssp_err_t	R_LPMV2_ClearIOKeep (void) Clear the IOKEEP bit after deep software stand by. More...

Detailed Description

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCU's.

Enumeration Type Documentation

◆ lpmv2_deep_standby_cancel_edge_t

enum lpmv2_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15-DS Pin Rising Edge (S7G2).

_RISING	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_deep_standby_cancel_source_t

enum lpmv2_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9 (S5D3, S5D5, S5D9, S7G2).

LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13 (S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14 (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15 (S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2 (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume (S5D3, S5D5, S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume (S5D9, S7G2).
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow (S5D3, S5D5, S5D9, S7G2).

◆ lpmv2_io_port_t

enum lpmv2_io_port_t	
I/O port state after Deep Software Standby mode (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state.
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode.

◆ lpmv2_output_port_enable_t

enum lpmv2_output_port_enable_t	
Output port enable (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpmv2_power_supply_t

enum lpmv2_power_supply_t	
Power supply control (S5D3, S5D5, S5D9, S7G2)	
Enumerator	
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode.
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled.

◆ lpmv2_snooze_cancel_t

enum lpmv2_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPM	ADC Channel 1 window compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt (all Synergy MCUs).
LPMV2_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt (all Synergy MCUs).

◆ lpmv2_snooze_dtc_t

enum lpmv2_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation (all Synergy MCUs).
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation (all Synergy MCUs).

◆ lpmv2_snooze_end_t

enum lpmv2_snooze_end_t	
Snooze end control	
Enumerator	
LPMV2_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch (S5D3)
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly (all Synergy MCUs).
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch (all Synergy MCUs).
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2)

◆ lpmv2_snooze_request_t

enum lpmv2_snooze_request_t	
Snooze request sources	
Enumerator	
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request (S124, S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2). Enable RXD0/DALI falling edge snooze request (S128, S1JA).
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).

LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request (S5D3, S5D5, S5D9, S7G2).
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request (all Synergy MCUs).
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request (all Synergy MCUs).

◆ lpmv2_standby_wake_source_t

enum lpmv2_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt (S124, S128, S1JA, S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt (S3A1, S3A3, S3A6, S3A7).
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7 (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12 (S3A1, S3A3, S3A6, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13 (S3A1, S3A3, S3A7, S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).

LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15 (S3A1, S3A3, S3A6, S3A7, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt (S5D3, S5D5, S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt (S5D9, S7G2).
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt (all Synergy MCUs).
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt (all Synergy MCUs).

Function Documentation

◆ R_LPMV2_ClearIOKeep()

ssp_err_t R_LPMV2_ClearIOKeep (void)

Clear the IOKEEP bit after deep software stand by.

Return values

SSP_SUCCESS	DSPBYCR_b.IOKEEP bit cleared Successfully.
SSP_ERR_UNSUPPORTED	Deep stand by mode not supported on this MCU.

◆ R_LPMV2_Init()

ssp_err_t R_LPMV2_Init (void)

Perform any necessary initialization.

Return values

SSP_SUCCESS	LPMV2 Software lock initialized

◆ R_LPMV2_LowPowerConfigure()

ssp_err_t R_LPMV2_LowPowerConfigure (lpmv2_cfg_t const *const *p_cfg*)

Configure a low power mode.

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

SSP_SUCCESS	Low power mode successfully applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

Get hardware lock

Unlock LPMV2 registers

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Configure MCU specific settings related to low power modes

Lock LPMV2 registers

Release hardware lock.

◆ R_LPMV2_LowPowerModeEnter()

ssp_err_t R_LPMV2_LowPowerModeEnter (void)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

Get hardware lock

Check for ongoing operating mode transition (OPCMTSF, SOPCMTSF)

Enter low power mode

Release hardware lock.

◆ R_LPMV2_VersionGet()

ssp_err_t R_LPMV2_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

Build Time Configurations

Renesas Synergy Software Package Reference » HAL Layer » LPMV2 S7G2

Macros

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)

Detailed Description

Macro Definition Documentation

◆ LPMV2_CFG_PARAM_CHECKING_ENABLE

#define LPMV2_CFG_PARAM_CHECKING_ENABLE (BSP_CFG_PARAM_CHECKING_ENABLE)
Specify whether to include code for API parameter checking. Valid settings include: • BSP_CFG_PARAM_CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h • 1 : Includes parameter checking • 0 : Compiles out parameter checking

5.1.5.41 LVD

Renesas Synergy Software Package Reference » HAL Layer

Driver for Low Voltage Detection. More...

Data Structures

struct	lvd_instance_ctrl_t
struct	lvd_extend_t

Enumerations

enum	lvd_sample_clock_t { LVD_SAMPLE_CLOCK_LOCO_DIV_2 = 0, LVD_SAMPLE_CLOCK_LOCO_DIV_4 = 1, LVD_SAMPLE_CLOCK_LOCO_DIV_8 = 2, LVD_SAMPLE_CLOCK_LOCO_DIV_16 = 3, LVD_SAMPLE_CLOCK_DISABLED = -1 }
enum	lvd_negation_delay_t { LVD_NEGATION_DELAY_FROM_VOLTAGE = 0, LVD_NEGATION_DELAY_FROM_RESET = 1 }

Functions

ssp_err_t	R_LVD_Open (lvd_ctrl_t *const p_api_ctrl, lvd_cfg_t const *const p_cfg) Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure. More...
ssp_err_t	R_LVD_Close (lvd_ctrl_t *const p_api_ctrl) Disables the LVD peripheral. Closes the driver instance. More...

ssp_err_t	R_LVD_StatusGet (lvd_ctrl_t *const p_api_ctrl, lvd_status_t *p_lvd_status) Get the current state of the monitor, (threshold crossing detected, voltage currently within range) More...
ssp_err_t	R_LVD_StatusClear (lvd_ctrl_t *const p_api_ctrl) Clears the latched status of the monitor. More...
ssp_err_t	R_LVD_VersionGet (ssp_version_t *const p_version) Returns the LVD driver version based on compile time macros. More...

Detailed Description

Driver for Low Voltage Detection.

Implementation of the LVD API.For a detailed description see the Low Voltage Detection Interface.

Summary

This module implements the following interface: Low Voltage Detection Interface.

Enumeration Type Documentation

◆ lvd_negation_delay_t

enum lvd_negation_delay_t	
Negation of LVD signal follows reset or voltage in range	
Enumerator	
LVD_NEGATION_DELAY_FROM_VOLTAGE	Negation follows a stabilization time (tLVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE
LVD_NEGATION_DELAY_FROM_RESET	Negation follows a stabilization time (tLVDn) after assertion of the LVDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE

◆ lvd_sample_clock_t

enum lvd_sample_clock_t	
Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering	
Enumerator	
LVD_SAMPLE_CLOCK_LOCO_DIV_2	Digital filter sample clock is LOCO divided by 2.
LVD_SAMPLE_CLOCK_LOCO_DIV_4	Digital filter sample clock is LOCO divided by 4.
LVD_SAMPLE_CLOCK_LOCO_DIV_8	Digital filter sample clock is LOCO divided by 8.
LVD_SAMPLE_CLOCK_LOCO_DIV_16	Digital filter sample clock is LOCO divided by 16.
LVD_SAMPLE_CLOCK_DISABLED	Digital filter is disabled.

Function Documentation

◆ R_LVD_Close()

ssp_err_t R_LVD_Close (lvd_ctrl_t *const *p_api_ctrl*)

Disables the LVD peripheral. Closes the driver instance.

Implements

- lvd_api_t::close.

Return values

SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	Pointers passed as function argument is NULL or the monitor number is invalid.
SSP_ERR_NOT_OPEN	Driver is not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::eventInfoGet

Disable voltage monitor comparison result output

Disable reset/interrupt event

Clear low voltage detection flag LVDnSR.DET = 0

Disable digital filtering

Disable voltage monitor

◆ R_LVD_Open()

ssp_err_t R_LVD_Open (lvd_ctrl_t *const *p_api_ctrl*, lvd_cfg_t const *const *p_cfg*)

Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.

Implements

- lvd_api_t::open.

Note

Digital filter is not to be used with standby modes

Return values

SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	One or more pointers passed as function argument point to NULL or the Requested configuration, detection, voltage, monitor number or sample clock configuration of a voltage monitor is invalid.
SSP_ERR_IN_USE	Unable to acquire the hardware lock.
SSP_ERR_INVALID_MODE	MOCO must be running for LVD_NEGATION_DELAY_FROM_RESET negation delay option

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Verify channel is not already used

Select the detection voltage by setting the LVDLVLR.LVDnLVL[m:0] bits.

Enable voltage detection LVCMPCR.LVDnE = 1

Configure the digital filter.

Configure LVD monitor detection response.

Enable output of the results of comparison by voltage monitor LVDnCR0.CMPE = 1

Mark driver as opened by initializing it to "LVD" in its ASCII equivalent.

◆ R_LVD_StatusClear()

ssp_err_t R_LVD_StatusClear (lvd_ctrl_t *const *p_api_ctrl*)

Clears the latched status of the monitor.

Implements

- lvd_api_t::statusClear.

Return values

SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	Pointers passed as function argument point to NULL, invalid LVD monitor number
SSP_ERR_NOT_OPEN	Driver is not open, status will not be valid

Clears the latched status of the monitor

◆ R_LVD_StatusGet()

ssp_err_t R_LVD_StatusGet (lvd_ctrl_t *const *p_api_ctrl*, lvd_status_t * *p_lvd_status*)

Get the current state of the monitor, (threshold crossing detected, voltage currently within range)

Implements

- lvd_api_t::statusGet.

Return values

SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	One or more pointers passed as function argument point to NULL, invalid LVD monitor number
SSP_ERR_NOT_OPEN	Driver is not open, status will not be valid

Get the current state of the monitor

◆ R_LVD_VersionGet()

ssp_err_t R_LVD_VersionGet (ssp_version_t *const *p_version*)
Returns the LVD driver version based on compile time macros. **Implements** ◦ lvd_api_t::versionGet. **Return values** SSP_SUCCESS — Successful SSP_ERR_ASSERTION — p_version was NULL

lvd_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » LVD

#include <r_lvd.h>

Data Fields

uint32_t	monitor_number Monitor number. More...
R_SYSTEM_Type *	p_reg Pointer to LVD register base address.
void(*	p_callback)(lvd_callback_args_t *p_args) Pointer to user callback.
lvd_callback_args_t	lvd_callback_args LVD callback parameters arguments.
uint32_t	opened Whether or not channel is open.

Detailed Description

LVD instance control structure

Field Documentation

◆ monitor_number

uint32_t lvd_instance_ctrl_t::monitor_number
Monitor number. Monitor number, 1, 2, ...

The documentation for this struct was generated from the following file:

- r_lvd.h

lvd_extend_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » LVD

#include <r_lvd.h>

Data Fields

lvd_negation_delay_t	negation_delay
lvd_sample_clock_t	sample_clock_divisor

Detailed Description

Hardware extend structure

Field Documentation

◆ negation_delay

lvd_negation_delay_t lvd_extend_t::negation_delay
Negation of LVD signal follows reset or voltage in range

◆ sample_clock_divisor

lvd_sample_clock_t lvd_extend_t::sample_clock_divisor
Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

---

The documentation for this struct was generated from the following file:

- r_lvd.h

5.1.5.42 Operational Amplifier (OPAMP)

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Operational Amplifier (OPAMP). More...

Data Structures

struct	opamp_on_opamp_cfg_t
struct	opamp_instance_ctrl_t

Macros

#define	OPAMP_CODE_VERSION_MAJOR (2U)
#define	OPAMP_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_opamp_version)

Enumerations

enum	opamp_trigger_t { OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP = 0, OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP = 1, OPAMP_TRIGGER_AGT_START_ADC_STOP = 3 }
enum	opamp_agt_link_t { OPAMP_AGT_LINK_AGT1_OPAMP_0_2_AGT0_OPAMP_1_3 = 0, OPAMP_AGT_LINK_AGT1_OPAMP_0_1_AGT0_OPAMP_2_3 = 1, OPAMP_AGT_LINK_AGT1_OPAMP_0_1_2_3 = 3 }
enum	opamp_mode_t { OPAMP_MODE_LOW_POWER = 0, OPAMP_MODE_MIDDLE_SPEED = 1, OPAMP_MODE_HIGH_SPEED = 3 }
enum	opamp_priv_trim_state_t {

	OPAMP_PRIV_TRIM_STATE_INVALID = -1, OPAMP_PRIV_TRIM_STATE_END = 0, OPAMP_PRIV_TRIM_STATE_BIT_0 = 0, OPAMP_PRIV_TRIM_STATE_BIT_1 = 1, OPAMP_PRIV_TRIM_STATE_BIT_2 = 2, OPAMP_PRIV_TRIM_STATE_BIT_3 = 3, OPAMP_PRIV_TRIM_STATE_BIT_4 = 4, OPAMP_PRIV_TRIM_STATE_BEGIN = 5 }

Functions

ssp_err_t	R_OPAMP_Open (opamp_ctrl_t *const p_api_ctrl, opamp_cfg_t const *const p_cfg) Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements opamp_api_t::open(). More...
ssp_err_t	R_OPAMP_InfoGet (opamp_ctrl_t *const p_api_ctrl, opamp_info_t *const p_info) Provides the minimum stabilization wait time in microseconds. Implements opamp_api_t::infoGet(). More...
ssp_err_t	R_OPAMP_Start (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements opamp_api_t::start(). More...
ssp_err_t	R_OPAMP_Stop (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements opamp_api_t::stop(). More...
ssp_err_t	R_OPAMP_StatusGet (opamp_ctrl_t *const p_api_ctrl, opamp_status_t *const p_status) Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed. Implements opamp_api_t::statusGet(). More...
ssp_err_t	R_OPAMP_Trim (opamp_ctrl_t *const p_api_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args) On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values. More...

ssp_err_t	R_OPAMP_Close (opamp_ctrl_t *const p_api_ctrl) Stops the op-amps. Implements opamp_api_t::close(). More...
ssp_err_t	R_OPAMP_VersionGet (ssp_version_t *const p_version)

Variables

const opamp_api_t	g_opamp_on_opamp

Detailed Description

Driver for the Operational Amplifier (OPAMP).

This module supports the OPAMP peripheral. It implements the following interfaces:

- OPAMP Interface

Macro Definition Documentation

◆ OPAMP_CODE_VERSION_MAJOR

#define OPAMP_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

◆ OPAMP_ERROR_RETURN

#define OPAMP_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_opamp_version)
Macro for error logger.

Enumeration Type Documentation

◆ opamp_agt_link_t

enum opamp_agt_link_t	
Which AGT timer starts the op-amp. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel. If OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP is selected for a channel, then no AGT compare match event will start that op-amp channel.	
Enumerator	
OPAMP_AGT_LINK_AGT1_OPAMP_0_2_AGT0_OPAMP_1_3	OPAMP channel 0 and 2 are started by AGT1 compare match. OPAMP channel 1 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_AGT0_OPAMP_2_3	OPAMP channel 0 and 1 are started by AGT1 compare match. OPAMP channel 2 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_2_3	All OPAMP channels are started by AGT1 compare match.

◆ opamp_mode_t

enum opamp_mode_t	
Op-amp mode.	
Enumerator	
OPAMP_MODE_LOW_POWER	Low power mode.
OPAMP_MODE_MIDDLE_SPEED	Middle speed mode (not supported on all MCUs)
OPAMP_MODE_HIGH_SPEED	High speed mode.

◆ opamp_priv_trim_state_t

enum opamp_priv_trim_state_t	
Op-amp trim state.	
Enumerator	
OPAMP_PRIV_TRIM_STATE_INVALID	Trim state invalid.
OPAMP_PRIV_TRIM_STATE_END	Trim state end.
OPAMP_PRIV_TRIM_STATE_BIT_0	Trim state bit 0.
OPAMP_PRIV_TRIM_STATE_BIT_1	Trim state bit 1.
OPAMP_PRIV_TRIM_STATE_BIT_2	Trim state bit 2.
OPAMP_PRIV_TRIM_STATE_BIT_3	Trim state bit 3.
OPAMP_PRIV_TRIM_STATE_BIT_4	Trim state bit 4.
OPAMP_PRIV_TRIM_STATE_BEGIN	Trim state begin.

◆ opamp_trigger_t

enum opamp_trigger_t	
Start and stop trigger for the op-amp.	
Enumerator	
OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP	Start and stop with APIs.
OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP	Start by AGT compare match and stop with API.
OPAMP_TRIGGER_AGT_START_ADC_STOP	Start by AGT compare match and stop after ADC conversion.

Function Documentation

◆ R_OPAMP_Close()

ssp_err_t R_OPAMP_Close (opamp_ctrl_t *const *p_api_ctrl*)

Stops the op-amps. Implements opamp_api_t::close().

Return values

SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

Set all OPAMP units and the reference current generator to be stopped.

Mark driver as closed

Enter the module-stop state.

Release the hardware lock

Return the error code

◆ R_OPAMP_InfoGet()

ssp_err_t R_OPAMP_InfoGet (opamp_ctrl_t *const *p_api_ctrl*, opamp_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements opamp_api_t::infoGet().

Return values

SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

◆ R_OPAMP_Open()

ssp_err_t R_OPAMP_Open (opamp_ctrl_t *const *p_api_ctrl*, opamp_cfg_t const *const *p_cfg*)

Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements opamp_api_t::open().

The op-amp is not operational until the opamp_api_t::start() is called. If the op-amp is configured to start after AGT compare match, the op-amp is not operational until opamp_api_t::start() and the associated AGT compare match event occurs.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open() and before opamp_api_t::start().

Return values

SSP_SUCCESS	Configuration successful.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.
SSP_ERR_IN_USE	Control block is already opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Verify the control block has not already been initialized.

Confirm the OPAMP exists on this MCU.

Lock the OPAMP

Initialize the hardware based on the configuration.

Stop operation of all op-amps.

Initialize trim registers to factory trim values.

◆ R_OPAMP_Start()

ssp_err_t R_OPAMP_Start (opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask*)

If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements opamp_api_t::start().

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open() and before opamp_api_t::start().

Return values

SSP_SUCCESS	Op-amp started or hardware triggers enabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

Enable AGT start and ADC conversion end triggers or start the op-amp channel(s).

Return the error code

◆ R_OPAMP_StatusGet()

ssp_err_t R_OPAMP_StatusGet (opamp_ctrl_t *const *p_api_ctrl*, opamp_status_t *const *p_status*)

Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed. Implements opamp_api_t::statusGet().

Return values

SSP_SUCCESS	Operating status of each op-amp provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Read the operating status of the op-amps.

◆ R_OPAMP_Stop()

ssp_err_t R_OPAMP_Stop (opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask*)

Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements opamp_api_t::stop().

Return values

SSP_SUCCESS	Op-amp stopped or hardware triggers disabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

Disable AGT start and ADC conversion end triggers and stop the op-amp channel(s).

Return the error code

◆ R_OPAMP_Trim()

ssp_err_t R_OPAMP_Trim (opamp_ctrl_t *const *p_api_ctrl*, opamp_trim_cmd_t const *cmd*, opamp_trim_args_t const *const *p_args*)

On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Not supported on all MCUs. See hardware manual for details. Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling trim() with command OPAMP_TRIM_CMD_START again.

Implements opamp_api_t::trim().

The trim procedure works as follows:

1. Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
2. Connect a fixed voltage to the Pch (+) input.
3. Connect the Nch (-) input to the op-amp output to create a voltage follower.
4. Ensure the op-amp is operating and stabilized.
5. Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
6. Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
7. Iterate over the following loop 5 times:
    a. Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
    b. Measure the op-amp output using the SAR ADC (referred to as B in the next step).
    c. If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

8. Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
9. Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
10. Iterate over the following loop 5 times:
    a. Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
    b. Measure the op-amp output using the SAR ADC (referred to as B in the next step).
    c. If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

Return values

SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_UNSUPPORTED	Trimming is not supported on this MCU.
SSP_ERR_INVALID_STATE	The command is not valid in the current state of the trim state machine.
SSP_ERR_INVALID_ARGUMENT	The requested channel is not operating or the trim procedure is not in progress for this channel/input combination.
SSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Initialize the trim register to 0 during OPAMP_TRIM_CMD_START.

Set the next trim bit during OPAMP_TRIM_CMD_NEXT_STEP.

Clear the current trim bit during OPAMP_TRIM_CMD_CLEAR_BIT.

◆ R_OPAMP_VersionGet()

ssp_err_t R_OPAMP_VersionGet (ssp_version_t *const *p_version*)

Gets the API and code version. Implements opamp_api_t::versionGet().

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

Variable Documentation

◆ g_opamp_on_opamp

const opamp_api_t g_opamp_on_opamp

```
=
{
    .open                = R_OPAMP_Open,
    .start               = R_OPAMP_Start,
    .stop                = R_OPAMP_Stop,
    .trim                = R_OPAMP_Trim,
    .infoGet             = R_OPAMP_InfoGet,
    .statusGet           = R_OPAMP_StatusGet,
    .close               = R_OPAMP_Close,
    .versionGet          = R_OPAMP_VersionGet
}
```

Name of module used by error logger macro OPAMP Implementation of OPAMP interface.

opamp_on_opamp_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Operational Amplifier (OPAMP)

#include <r_opamp.h>

Data Fields

opamp_agt_link_t	agt_link
opamp_mode_t	mode Low power, middle speed, or high speed mode.
opamp_trigger_t	trigger_channel_0 Start and stop triggers for channel 0.
opamp_trigger_t	trigger_channel_1 Start and stop triggers for channel 1.

opamp_trigger_t	trigger_channel_2 Start and stop triggers for channel 2.
opamp_trigger_t	trigger_channel_3 Start and stop triggers for channel 3.

Detailed Description

OPAMP configuration extension. This extension is required and must be provided in opamp_cfg_t::p_extend.

Field Documentation

◆ agt_link

opamp_agt_link_t opamp_on_opamp_cfg_t::agt_link
Configure which AGT links are paired to which channel. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel.

The documentation for this struct was generated from the following file:

- r_opamp.h

opamp_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Operational Amplifier (OPAMP)

#include <r_opamp.h>

Data Fields

R_OPAMP_Type *	p_reg Base register for this unit.
uint32_t	opened Boolean to verify that the Unit has been initialized.

uint8_t	trim_capable OPAMP has trim registers.
uint8_t	switches OPAMP has switches.
uint32_t	valid_opamps Mask of valid op-amps.
opamp_priv_trim_state_t	trim_state Trim state for each op-amp.
uint8_t	trim_channel Channel.
opamp_trim_input_t	trim_input Which input of the channel above.

Detailed Description

OPAMP instance control block. DO NOT INITIALIZE. Initialized in opamp_api_t::open().

The documentation for this struct was generated from the following file:

- r_opamp.h

5.1.5.43 PDC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Parallel Data Capture Unit (PDC). More...

Data Structures

struct	pdc_instance_ctrl_t

Functions

ssp_err_t	R_PDC_Open (pdc_ctrl_t *const p_api_ctrl, pdc_cfg_t const *const p_cfg) Powers on PDC, handles required initialization described in the hardware manual. Implements pdc_api_t::open. More...
ssp_err_t	R_PDC_Close (pdc_ctrl_t *const p_api_ctrl) Stops and closes the transfer interface, disables the PDC, powers off the PDC, clears internal driver data and disables interrupts. Implements pdc_api_t::close. More...
ssp_err_t	R_PDC_CaptureStart (pdc_ctrl_t *const p_api_ctrl, uint8_t *const p_buffer) Starts a capture. Enables interrupts. Implements pdc_api_t::captureStart. More...
ssp_err_t	R_PDC_StateGet (pdc_ctrl_t *const p_api_ctrl, pdc_state_t *p_state) Returns the state of the VSYNC and HSYNC pins.Implements pdc_api_t::stateGet. More...
ssp_err_t	R_PDC_VersionGet (ssp_version_t *const p_data) Return PDC HAL driver version. Implements pdc_api_t::versionGet. More...

Detailed Description

Driver for the Parallel Data Capture Unit (PDC).

Summary

extends PDC Interface The PDC interface allows the capturing of an image from a camera module.

Function Documentation

◆ R_PDC_CaptureStart()

ssp_err_t R_PDC_CaptureStart (pdc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_buffer*)

Starts a capture. Enables interrupts. Implements pdc_api_t::captureStart.

Sets up the transfer interface to transfer data from the PDC into the specified buffer. Configures the PDC settings as previously set by the pdc_api_t::open API. These settings are configured here as the PIXCLK input must be active for the PDC reset operation. When a capture is complete the callback registered during pdc_api_t::open API call will be called.

Return values

SSP_SUCCESS	Capture start successful.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_api_ctrl is NULL, OR • low level transfer is not assigned, OR • low level transfer APIs are not assigned • buffer is not assigned, assign buffer
SSP_ERR_NOT_OPEN	Open has not been successfully called.
SSP_ERR_IN_USE	Pdc transfer is already in progress, wait for transfer to complete.
SSP_ERR_TIMEOUT	Reset operation timed out.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::open
- transfer_api_t::enable

Note

If the PIXCLK is being generated by a camera module the camera must be configured after the call to pdc_api_t::open and before the call to pdc_api_t::captureStart. This function is not reentrant.

The user is responsible to ensuring that the memory pointed to by p_buffer is both valid and large enough to store a complete image. The amount of space required, in bytes can be calculated as shown:

size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel

Set up transfer interface

Configure the transfer interface

Open transfer interface

Wait for reset to complete

Set horizontal capture range

Set horizontal capture size

Set vertical capture range

Set vertical capture size

Set VSYNC polarity

Set HSYNC polarity

Set endianess of capture data

Enable interrupts: Receive data ready interrupt, Underrun interrupt, Overrun interrupt, Frame end interrupt, Vertical line number setting error interrupt, Horizontal byte number setting error interrupt

◆ R_PDC_Close()

ssp_err_t R_PDC_Close (pdc_ctrl_t *const *p_api_ctrl*)

Stops and closes the transfer interface, disables the PDC, powers off the PDC, clears internal driver data and disables interrupts. Implements pdc_api_t::close.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_api_ctrl is NULL, OR • low level transfer is not assigned, OR • low level transfer APIs are not assigned
SSP_ERR_NOT_OPEN	Open has not been successfully called.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::disable
- transfer_api_t::close

Note

This API will close the PDC driver. If a capture is in progress it will be stopped. This function is reentrant.

Disable all interrupts.

Enable module stop mode for PDC

Unlock the PDC Hardware Resource

◆ R_PDC_Open()

ssp_err_t R_PDC_Open (pdc_ctrl_t *const *p_api_ctrl*, pdc_cfg_t const *const *p_cfg*)

Powers on PDC, handles required initialization described in the hardware manual. Implements pdc_api_t::open.

The Open function provides initial configuration for the PDC module. It powers on the module and enables the PCLKO output and the PIXCLK input. Further initialization requires the PIXCLK input to be running in order to be able to reset the PDC as part of its initialization. This clock is input from a camera module and so the reset and further initialization is performed in pdc_api_t::captureStart. This function should be called once prior to calling any other PDC API functions. After the PDC is opened the Open function should not be called again without first calling the Close function.

Return values

SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either

	• p_cfg is NULL, OR • p_api_ctrl is NULL, OR • The pointer to the transfer interface in the p_cfg parameter is NULL
SSP_ERR_INVALID_ARGUMENT	One of the following configuration parameters is incorrect. Either • bytes_per_pixel is zero, OR • x_capture_pixels is zero, OR • y_capture_pixels is zero, OR • x_capture_start_pixel + x_capture_pixels is greater than 4095, OR • y_capture_start_pixel + y_capture_pixels is greater than 4095
SSP_ERR_HW_LOCKED	Unable to reserve BSP hardware lock for this module.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is not reentrant.

Lock the PDC Hardware Resource

Disable module stop mode for PDC

Set PCLKB divider

Enable PCLKO output

Enable the PIXCLK input

Mark driver as open by initializing it to "PDC" in its ASCII equivalent

◆ R_PDC_StateGet()

ssp_err_t R_PDC_StateGet (pdc_ctrl_t *const *p_api_ctrl*, pdc_state_t * *p_state*)

Returns the state of the VSYNC and HSYNC pins.Implements pdc_api_t::stateGet.

Return values

SSP_SUCCESS	State read successful.
SSP_ERR_ASSERTION	p_api_ctrl is NULL OR p_state is NULL
SSP_ERR_NOT_OPEN	Open has not been successfully called.

Note

This function is reentrant.

Check if the driver is open

Get the contents of PCMONR register

Update vsync signal state

Update hsync signal state

◆ R_PDC_VersionGet()

ssp_err_t R_PDC_VersionGet (ssp_version_t *const *p_data*)

Return PDC HAL driver version. Implements pdc_api_t::versionGet.

Return values

SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter

Note

This function is reentrant.

pdc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » PDC

#include <r_pdc.h>

Data Fields

R_PDC_Type *	p_reg Pointer to PDC base address.

uint32_t	open
uint8_t	bytes_per_pixel Number of bytes per pixel.
uint16_t	x_resolution_pixels Total number of horizontal pixels input to PDC.
uint16_t	y_resolution_pixels Total number of lines input to the PDC.
uint16_t	x_capture_start_pixel Horizontal position to start capture.
uint16_t	x_capture_pixels Number of horizontal pixels to capture.
uint16_t	y_capture_start_pixel Vertical position to start capture.
uint16_t	y_capture_pixels Number of vertical lines/pixels to capture.
pdc_endian_t	endian Endian of capture data.
pdc_hsync_polarity_t	hsync_polarity Polarity of HSYNC input.
pdc_vsync_polarity_t	vsync_polarity Polarity of VSYNC input.
uint8_t *	p_current_buffer Pointer to buffer currently in use.

bool	transfer_in_progress Indicates if a PDC transfer is already in progress.
transfer_instance_t const *	p_lower_lvl_transfer Pointer to the transfer instance the PDC should use.
transfer_info_t	info_transfer Transfer info structure for low level Transfer interface.
void const *	p_context
void(*	p_callback)(pdc_callback_args_t *p_args) Callback provided when a PDC transfer ISR occurs.
IRQn_Type	frame_end_irq Frame end interrupt number.
IRQn_Type	irq PDC interrupt number.

Detailed Description

PDC instance control block. DO NOT INITIALIZE. Initialization occurs when pdc_api_t::open is called.

Field Documentation

◆ open

uint32_t pdc_instance_ctrl_t::open
Indicates whether or not the driver is open called.

◆ p_context

void const* pdc_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in pdc_callback_args_t.

The documentation for this struct was generated from the following file:

- r_pdc.h

5.1.5.44 PTP

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Precision time protocol(PTP). More...

Data Structures

struct	ptp_instance_ctrl_t

Functions

ssp_err_t	R_PTP_Open (ptp_ctrl_t *const p_api_ctrl, ptp_cfg_t const *const p_cfg) Open the PTP driver, handles required initialization described in hardware manual. Implements ptp_api_t::open. More...
ssp_err_t	R_PTP_Close (ptp_ctrl_t *const p_api_ctrl) Closes the PTP driver. Implements ptp_api_t::close. More...
ssp_err_t	R_PTP_Configure (ptp_ctrl_t *const p_api_ctrl, uint32_t *p_ip_address, uint32_t *p_physical_address_msw, uint32_t *p_physical_address_lsw) Configures the PTP driver with IP and MAC address. Implements ptp_api_t::configure. More...
ssp_err_t	R_PTP_SetExtPromiscuous (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, bool is_set) Sets or clear the extended promiscuous mode of the specified PTP channel. Implements ptp_api_t::setExtPromiscuous. More...
ssp_err_t	R_PTP_GetLocalClock (ptp_ctrl_t *const p_api_ctrl, ptp_timestamp_t *p_clock, uint32_t wait_option) Gets the current local clock counter value in Timestamp format(UNIX time) when configured as slave. Implements ptp_api_t::getLocalClock . More...

ssp_err_t	R_PTP_SetLocalClock (ptp_ctrl_t *const p_api_ctrl, ptp_timestamp_t *p_clock) Sets local clock counter value with Timestamp (UNIX time). Master clock set the master time. Implements ptp_api_t::setLocalClock. More...
ssp_err_t	R_PTP_GetMasterPortID (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port) Gets master clock ID and master port number fields of the specified PTP channel. Note: If the argument (p_clock or p_port) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getMasterPortID. More...
ssp_err_t	R_PTP_SetMasterPortID (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, uint32_t *p_clock, uint16_t *p_port) Sets master clock ID and master port number fields of the specified PTP channel. Note: If the argument (p_clock or p_port) is NULL pointer, the value will not be updated. Implements ptp_api_t::setMasterPortID. More...
ssp_err_t	R_PTP_GetSyncInfo (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, ptp_timeInterval_t *p_master_offset, ptp_timeInterval_t *p_path_delay) Gets the current offset from master and mean path delay of the specified PTP channel when configured as slave. Note: If the argument (p_master_offset or p_path_delay) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getSyncInfo. More...
ssp_err_t	R_PTP_Start (ptp_ctrl_t *const p_api_ctrl, uint32_t wait_option) Starts time synchronization. Implements ptp_api_t::start. More...
ssp_err_t	R_PTP_Stop (ptp_ctrl_t *const p_api_ctrl, uint32_t wait_option) Stops time synchronization. Implements ptp_api_t::stop. More...
ssp_err_t	R_PTP_SetWorst10Values (ptp_ctrl_t *const p_api_ctrl, uint8_t interval) Sets the interval for collecting worst 10 values. Implements ptp_api_t::setWorst10Values. More...
ssp_err_t	R_PTP_CheckWorst10Values (ptp_ctrl_t *const p_api_ctrl, uint32_t

	wait_option) Checks if worst 10 values are acquired and set as gradient limits when configured as slave. Implements ptp_api_t::checkWorst10Values. More...
ssp_err_t	R_PTP_GetWorst10Values (ptp_ctrl_t *const p_api_ctrl, uint32_t *p_positive_worst10, uint32_t *p_negative_worst10, uint32_t wait_option) Gets positive and negative worst 10 values by software. Note: If the argument (p_positive_worst10 or p_negative_worst10) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getWorst10Values. More...
ssp_err_t	R_PTP_SetGradientLimit (ptp_ctrl_t *const p_api_ctrl, uint32_t *p_positive_limit, uint32_t *p_negative_limit) Sets the gradient limits for positive and negative worst 10 values. Note: If the argument (p_positive_limit or p_negative_limit) is NULL pointer, the value will not be acquired. Implements ptp_api_t::setGradientLimit. More...
ssp_err_t	R_PTP_UpdateClockID (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, int8_t *p_clock_id) Updates clock identity field. Implements ptp_api_t::updateClockID. More...
ssp_err_t	R_PTP_UpdateDomainNumber (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, uint8_t domain_num) Updates domain number field in the message header. Implements ptp_api_t::updateDomainNumber. More...
ssp_err_t	R_PTP_UpdateAnnounceFlags (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, ptp_announce_flag_t *p_flag) Updates announce message's flag field. Implements ptp_api_t::updateAnnounceFlags. More...
ssp_err_t	R_PTP_UpdateAnnounceMsgs (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, ptp_announce_message_t *p_message) Updates announce message's message field. Implements ptp_api_t::updateAnnounceMsgs. More...
ssp_err_t	R_PTP_UpdateSyncAnnounceMsgInterval (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, int8_t *p_sync_interval, int8_t

	*p_announce_interval) Updates transmission interval and logMessageInterval of Sync and Announce messages. Note: If the argument (p_sync_interval or p_announce_interval) is NULL pointer, the value will not be acquired. Implements ptp_api_t::updateSyncAnnounceMsgInterval. More...
ssp_err_t	R_PTP_UpdateDelayMsgInterval (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, int8_t *p_interval, uint32_t *p_timeout) Updates transmission interval, logMessageInterval and timeout values of Delay message. Note: If the argument (p_interval or p_timeout) is NULL pointer, the value will not be updated. Implements ptp_api_t::updateDelayMsgInterval. More...
ssp_err_t	R_PTP_GetMessageReceptionConfig (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception) Gets PTP message reception synchronous configuration. Implements ptp_api_t::getMessageReceptionConfig. More...
ssp_err_t	R_PTP_SetMessageReceptionConfig (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, ptp_message_reception_t *p_ptp_message_reception) Sets PTP message reception synchronous configuration. Implements ptp_api_t::setMessageReceptionConfig. More...
ssp_err_t	R_PTP_DisableTimer (ptp_ctrl_t *const p_api_ctrl, ptp_stca_timer_channel_t timer_channel) Disables the specified timer event interrupt. Implements ptp_api_t::disableTimer. More...
ssp_err_t	R_PTP_IndicateEvent (ptp_ctrl_t *const p_api_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set) Set/clear interrupt indication to ELC output on generation of pulse produced by pulse output timer. Implements ptp_api_t::indicateEvent . More...
ssp_err_t	R_PTP_AutoClearEvent (ptp_ctrl_t *const p_api_ctrl, ptp_stca_timer_channel_t timer_channel, ptp_stca_timer_pulse_edge_t timer_pulse_edge, bool is_set) Set/clear auto clear mode for enabling one time output of ELC event. Implements ptp_api_t::autoClearEvent. More...

ssp_err_t	R_PTP_SetTimer (ptp_ctrl_t *const p_api_ctrl, uint8_t timer_channel, UInt64_t event_time, uint32_t cycle, uint32_t pulse_width) Sets start time, pulse period and pulse width for the pulse output timer. Implements ptp_api_t::setTimer. More...
ssp_err_t	R_PTP_SetMINTevent (ptp_ctrl_t *const p_api_ctrl, ptp_event_t ptp_reg, uint32_t event, bool is_set) Sets MINT interrupt event to enable notification for change in state of modules Implements ptp_api_t::setMINTevent. More...
ssp_err_t	R_PTP_EnableINFABTnotification (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel) Enables EPTPC INFABT notification of the specified PTP channel. Implements ptp_api_t::enableINFABTnotification. More...
ssp_err_t	R_PTP_DisableINFABTnotification (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel) Disables EPTPC INFABT notification of the specified PTP channel. Implements ptp_api_t::disableINFABTnotification. More...
ssp_err_t	R_PTP_CheckINFABTstatus (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel, uint8_t *p_status) Checks the status of INFABT flag of the specified PTP channel. Implements ptp_api_t::checkINFABTstatus. More...
ssp_err_t	R_PTP_ClearINFABTstatus (ptp_ctrl_t *const p_api_ctrl, uint8_t ptp_channel) Clears INFABT interrupt occurrence flag of the specified PTP channel. Implements ptp_api_t::clearINFABTstatus. More...
ssp_err_t	R_PTP_VersionGet (ssp_version_t *const p_version) Gets version information and stores it in the provided version struct. Implements ptp_api_t::versionGet. More...

Detailed Description

Driver for the Precision time protocol(PTP).

Summary

This module supports the PTP time synchronization functionality.

Extends PTP driver Interface.

Function Documentation

◆ R_PTP_AutoClearEvent()

ssp_err_t R_PTP_AutoClearEvent (ptp_ctrl_t *const *p_api_ctrl*, ptp_stca_timer_channel_t *timer_channel*, ptp_stca_timer_pulse_edge_t *timer_pulse_edge*, bool *is_set*)

Set/clear auto clear mode for enabling one time output of ELC event. Implements ptp_api_t::autoClearEvent.

Return values

SSP_SUCCESS	ELC interrupt auto clear mode set or clear is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Get IPLS Interrupt Permission Automatic clearing register status

If set, enable automatic clearing for specified pulse output timer channel

Set IPLS Interrupt Permission Automatic clearing register status

◆ R_PTP_CheckINFABTstatus()

ssp_err_t R_PTP_CheckINFABTstatus (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, uint8_t * *p_status*)

Checks the status of INFABT flag of the specified PTP channel. Implements ptp_api_t::checkINFABTstatus.

Return values

SSP_SUCCESS	INFABT flag status check is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Gets the current status of INFABT notification flag of the specified PTP channel

◆ R_PTP_CheckWorst10Values()

ssp_err_t R_PTP_CheckWorst10Values (ptp_ctrl_t *const *p_api_ctrl*, uint32_t *wait_option*)

Checks if worst 10 values are acquired and set as gradient limits when configured as slave. Implements ptp_api_t::checkWorst10Values.

Return values

SSP_SUCCESS	Gradient values are set based on worst 10 values successfully.
SSP_ERR_TIMEOUT	Gradient values are not set before timeout.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Wait till worst10 values are acquired by hardware and set as gradient limits

◆ R_PTP_ClearINFABTstatus()

ssp_err_t R_PTP_ClearINFABTstatus (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*)

Clears INFABT interrupt occurrence flag of the specified PTP channel. Implements ptp_api_t::clearINFABTstatus.

Return values

SSP_SUCCESS	INFABT status flag clear is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Clear the status of INFABT notification flag of the specified PTP channel

◆ R_PTP_Close()

ssp_err_t R_PTP_Close (ptp_ctrl_t *const *p_api_ctrl*)

Closes the PTP driver. Implements ptp_api_t::close.

Return values

SSP_SUCCESS	The PTP driver is successfully closed.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Disable PTP MINT interrupt

Release hardware lock

Mark driver as closed

◆ R_PTP_Configure()

ssp_err_t R_PTP_Configure (ptp_ctrl_t *const *p_api_ctrl*, uint32_t * *p_ip_address*, uint32_t * *p_physical_address_msw*, uint32_t * *p_physical_address_lsw*)

Configures the PTP driver with IP and MAC address. Implements ptp_api_t::configure.

Return values

SSP_SUCCESS	The PTP driver is successfully configured with IP and MAC address.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block or passed argument is NULL.

Initialize Synchronization Frame Processing unit (SYNFP)

Initialize Packet Relation Controller (PRC-TC) and Statistical Time Correction Algorithm (STCA) units

◆ R_PTP_DisableINFABTnotification()

ssp_err_t R_PTP_DisableINFABTnotification (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*)

Disables EPTPC INFABT notification of the specified PTP channel. Implements ptp_api_t::disableINFABTnotification.

Return values

SSP_SUCCESS	INFABT notification disable is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Disable INFABT detection flag of the specified PTP channel

◆ R_PTP_DisableTimer()

ssp_err_t R_PTP_DisableTimer (ptp_ctrl_t *const *p_api_ctrl*, ptp_stca_timer_channel_t *timer_channel*)

Disables the specified timer event interrupt. Implements ptp_api_t::disableTimer.

Return values

SSP_SUCCESS	Timer event interrupt disable is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Get MINT interrupt request status

Disable the timer event of specified pulse output channel

◆ R_PTP_EnableINFABTnotification()

ssp_err_t R_PTP_EnableINFABTnotification (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*)

Enables EPTPC INFABT notification of the specified PTP channel. Implements ptp_api_t::enableINFABTnotification.

Return values

SSP_SUCCESS	INFABT notification enable is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Initially clear the status of INFABT detection flag of the specified PTP channel

Enable generation of ETHER_MINT interrupt by SYNFP0 status flag

Enable generation of ETHER_MINT interrupt by SYNFP1 status flag

Enable the INFABT detection flag of specified PTP channel

◆ R_PTP_GetLocalClock()

ssp_err_t R_PTP_GetLocalClock (ptp_ctrl_t *const *p_api_ctrl*, ptp_timestamp_t * *p_clock*, uint32_t *wait_option*)

Gets the current local clock counter value in Timestamp format(UNIX time) when configured as slave. Implements ptp_api_t::getLocalClock.

Return values

SSP_SUCCESS	Local clock counter get is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_TIMEOUT	Clock info is not acquired before timeout.
SSP_ERR_ASSERTION	Pointer to the control block or p_clock parameter is NULL.

Request the current local clock counter information

Wait till local clock counter value is loaded with current local time

Save current local clock counter value

◆ R_PTP_GetMasterPortID()

ssp_err_t R_PTP_GetMasterPortID (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, uint32_t * *p_clock*, uint16_t * *p_port*)

Gets master clock ID and master port number fields of the specified PTP channel. Note: If the argument (p_clock or p_port) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getMasterPortID.

Return values

SSP_SUCCESS	Master port ID get is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block, p_clock or p_port parameter is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Get master clock ID high and low fields

Get master port number field

◆ R_PTP_GetMessageReceptionConfig()

ssp_err_t R_PTP_GetMessageReceptionConfig (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, ptp_message_reception_t * *p_ptp_message_reception*)

Gets PTP message reception synchronous configuration. Implements ptp_api_t::getMessageReceptionConfig.

Return values

SSP_SUCCESS	Synchronous configuration get is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Get SYNFP Reception Filter Register 1

Get SYNFP Reception Filter Register 2

Get SYNFP Transmission Enable Register

Get SYNFP Operation Setting Register

◆ R_PTP_GetSyncInfo()

ssp_err_t R_PTP_GetSyncInfo (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, ptp_timeInterval_t * *p_master_offset*, ptp_timeInterval_t * *p_path_delay*)

Gets the current offset from master and mean path delay of the specified PTP channel when configured as slave. Note: If the argument (p_master_offset or p_path_delay) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getSyncInfo.

Return values

SSP_SUCCESS	Offset from master and mean path delay get is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block, p_master_offset or p_path_delay is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Request offset from master from the specified PTP channel

Get the offset from master in nano-seconds format

Request mean path delay from the specified PTP channel

Get the mean path delay in nano-seconds format

◆ R_PTP_GetWorst10Values()

ssp_err_t R_PTP_GetWorst10Values (ptp_ctrl_t *const *p_api_ctrl*, uint32_t * *p_positive_worst10*, uint32_t * *p_negative_worst10*, uint32_t *wait_option*)

Gets positive and negative worst 10 values by software. Note: If the argument (p_positive_worst10 or p_negative_worst10) is NULL pointer, the value will not be acquired. Implements ptp_api_t::getWorst10Values.

Return values

SSP_SUCCESS	Positive and negative worst 10 values get is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Request current worst10 values acquired

Wait till gradient worst10 values are acquired by software

Get positive gradient values

Get negative gradient values

◆ R_PTP_IndicateEvent()

ssp_err_t R_PTP_IndicateEvent (ptp_ctrl_t *const *p_api_ctrl*, ptp_stca_timer_channel_t *timer_channel*, ptp_stca_timer_pulse_edge_t *timer_pulse_edge*, bool *is_set*)

Set/clear interrupt indication to ELC output on generation of pulse produced by pulse output timer. Implements ptp_api_t::indicateEvent.

Return values

SSP_SUCCESS	ELC interrupt indication set or clear is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Get IPLS Interrupt Permission Register status

If set, enable output from pulse output timer channel

Set IPLS Interrupt Permission Register status

◆ R_PTP_Open()

ssp_err_t R_PTP_Open (ptp_ctrl_t *const *p_api_ctrl*, ptp_cfg_t const *const *p_cfg*)

Open the PTP driver, handles required initialization described in hardware manual. Implements ptp_api_t::open.

Return values

SSP_SUCCESS	PTP driver has opened successfully and initialization was successful.
SSP_ERR_IN_USE	The channel specified has already been opened.
SSP_ERR_ASSERTION	The p_ctrl or p_cfg parameter was NULL.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Verify if this unit has not already been initialized

Make sure the peripheral exists

Lock EPTPC channel

Power on the PTP module.

Configure MINT interrupt

Reset PTP driver

Wait at least 64 cycles of PCLKA to reset the PTPEDMAC and EPTPC. PCLKA must be at least 12.5 MHz to use Ethernet, so wait at least 5.12 us.

Initialize the channel state information

Select synchronization frame processing unit (SYNFP0 or SYNFP1)

Initialize EPTPC MINT interrupt requests

Initialize SYNFP reception status

Enable MINT interrupt

Disable bypassing of EPTPC module

Mark driver as opened by initializing it to "PTP" in its ASCII equivalent for this unit

◆ R_PTP_SetExtPromiscuous()

ssp_err_t R_PTP_SetExtPromiscuous (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, bool *is_set*)

Sets or clear the extended promiscuous mode of the specified PTP channel. Implements ptp_api_t::setExtPromiscuous.

Return values

SSP_SUCCESS	Extended promiscuous mode is set/cleared successfully.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Set or clear extended promiscuous mode of the specified PTP channel

◆ R_PTP_SetGradientLimit()

ssp_err_t R_PTP_SetGradientLimit (ptp_ctrl_t *const *p_api_ctrl*, uint32_t * *p_positive_limit*, uint32_t * *p_negative_limit*)

Sets the gradient limits for positive and negative worst 10 values. Note: If the argument (p_positive_limit or p_negative_limit) is NULL pointer, the value will not be acquired. Implements ptp_api_t::setGradientLimit.

Return values

SSP_SUCCESS	Positive and negative gradient values set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Set positive gradient values

Set negative gradient values

◆ R_PTP_SetLocalClock()

ssp_err_t R_PTP_SetLocalClock (ptp_ctrl_t *const *p_api_ctrl*, ptp_timestamp_t * *p_clock*)

Sets local clock counter value with Timestamp (UNIX time). Master clock set the master time. Implements ptp_api_t::setLocalClock.

Return values

SSP_SUCCESS	Local clock counter set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block or p_clock parameter is NULL.

Set local clock counter value with the specified time information

◆ R_PTP_SetMasterPortID()

ssp_err_t R_PTP_SetMasterPortID (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, uint32_t * *p_clock*, uint16_t * *p_port*)

Sets master clock ID and master port number fields of the specified PTP channel. Note: If the argument (p_clock or p_port) is NULL pointer, the value will not be updated. Implements ptp_api_t::setMasterPortID.

Return values

SSP_SUCCESS	Master port ID is updated successfully.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block, p_clock or p_port is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Set master clock ID high and low fields

Set master port number field

◆ R_PTP_SetMessageReceptionConfig()

ssp_err_t R_PTP_SetMessageReceptionConfig (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, ptp_message_reception_t * *p_ptp_message_reception*)

Sets PTP message reception synchronous configuration. Implements ptp_api_t::setMessageReceptionConfig.

Return values

SSP_SUCCESS	Synchronous configuration set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Set SYNFP Reception Filter Register 1

Set SYNFP Reception Filter Register 2

Set SYNFP Transmission Enable Register

Set SYNFP Operation Setting Register

◆ R_PTP_SetMINTevent()

ssp_err_t R_PTP_SetMINTevent (ptp_ctrl_t *const *p_api_ctrl*, ptp_event_t *ptp_reg*, uint32_t *event*, bool *is_set*)

Sets MINT interrupt event to enable notification for change in state of modules Implements ptp_api_t::setMINTevent.

Return values

SSP_SUCCESS	MINT interrupt event set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Enable notification of STCA status

Enable notification of PRC-TC status

Enable notification of SYNFP0 status

Enable notification of SYNFP1 status

◆ R_PTP_SetTimer()

ssp_err_t R_PTP_SetTimer (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *timer_channel*, UInt64_t *event_time*, uint32_t *cycle*, uint32_t *pulse_width*)

Sets start time, pulse period and pulse width for the pulse output timer. Implements ptp_api_t::setTimer.

Return values

SSP_SUCCESS	Timer event set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Set event time, PWM cycle interval and PWM pulse width to specified timer channel

Start the specified pulse output timer

◆ R_PTP_SetWorst10Values()

ssp_err_t R_PTP_SetWorst10Values (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *interval*)

Sets the interval for collecting worst 10 values. Implements ptp_api_t::setWorst10Values.

Return values

SSP_SUCCESS	Worst 10 values are recorded successfully.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Sets interval to get worst 10 values

◆ R_PTP_Start()

ssp_err_t R_PTP_Start (ptp_ctrl_t *const *p_api_ctrl*, uint32_t *wait_option*)

Starts time synchronization. Implements ptp_api_t::start.

Return values

SSP_SUCCESS	Time synchronization started successfully.
SSP_ERR_TIMEOUT	Time synchronization did not start before timeout.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_MODE	Invalid PTP mode.

Starts time synchronization

◆ R_PTP_Stop()

ssp_err_t R_PTP_Stop (ptp_ctrl_t *const *p_api_ctrl*, uint32_t *wait_option*)

Stops time synchronization. Implements ptp_api_t::stop.

Return values

SSP_SUCCESS	Time synchronization stopped successfully.
SSP_ERR_TIMEOUT	Time synchronization did not stop before timeout.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_MODE	Invalid PTP mode.

Stops the time synchronization

◆ R_PTP_UpdateAnnounceFlags()

ssp_err_t R_PTP_UpdateAnnounceFlags (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, ptp_announce_flag_t * *p_flag*)

Updates announce message's flag field. Implements ptp_api_t::updateAnnounceFlags.

Return values

SSP_SUCCESS	Announce message flag field set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Update announce message's flag field

◆ R_PTP_UpdateAnnounceMsgs()

ssp_err_t R_PTP_UpdateAnnounceMsgs (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, ptp_announce_message_t * *p_message*)

Updates announce message's message field. Implements ptp_api_t::updateAnnounceMsgs.

Return values

SSP_SUCCESS	Announce message field set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Update grandmasterPriority1 and grandmasterPriority2 fields

Update grandmasterClockQuality fields

Update grandmasterIdentity fields of Announce messages

◆ R_PTP_UpdateClockID()

ssp_err_t R_PTP_UpdateClockID (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, int8_t * *p_clock_id*)

Updates clock identity field. Implements ptp_api_t::updateClockID.

Return values

SSP_SUCCESS	Clock identity field set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Update local clockIdentity field of the specified EPTPC port

◆ R_PTP_UpdateDelayMsgInterval()

ssp_err_t R_PTP_UpdateDelayMsgInterval (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, int8_t * *p_interval*, uint32_t * *p_timeout*)

Updates transmission interval, logMessageInterval and timeout values of Delay message. Note: If the argument (p_interval or p_timeout) is NULL pointer, the value will not be updated. Implements ptp_api_t::updateDelayMsgInterval.

Return values

SSP_SUCCESS	Transmission interval, logMessage interval and timeout set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

If clock state is master, update Delay_Response logMessageInterval value

If clock state is slave, update Delay_Request /Pdelay_Request transmission interval value

Update Delay_Response/Pdelay_Response receiving timeout value

◆ R_PTP_UpdateDomainNumber()

ssp_err_t R_PTP_UpdateDomainNumber (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, uint8_t *domain_num*)

Updates domain number field in the message header. Implements ptp_api_t::updateDomainNumber.

Return values

SSP_SUCCESS	Domain number field set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Update domainNumber field of the PTP message header

◆ R_PTP_UpdateSyncAnnounceMsgInterval()

ssp_err_t R_PTP_UpdateSyncAnnounceMsgInterval (ptp_ctrl_t *const *p_api_ctrl*, uint8_t *ptp_channel*, int8_t * *p_sync_interval*, int8_t * *p_announce_interval*)

Updates transmission interval and logMessageInterval of Sync and Announce messages. Note: If the argument (p_sync_interval or p_announce_interval) is NULL pointer, the value will not be acquired. Implements ptp_api_t::updateSyncAnnounceMsgInterval.

Return values

SSP_SUCCESS	Transmission interval and logMessage interval set is successful.
SSP_ERR_NOT_OPEN	The PTP driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_CHANNEL	Invalid EPTPC channel.

Update SYNFP Sync message transmission interval

Update SYNFP announce message Transmission Interval

◆ R_PTP_VersionGet()

ssp_err_t R_PTP_VersionGet (ssp_version_t *const *p_version*)

Gets version information and stores it in the provided version struct. Implements ptp_api_t::versionGet.

Return values

SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was NULL.

Return the version number

ptp_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » PTP

#include <r_ptp.h>

Data Fields

ssp_err_t(*	p_callback)(ptp_callback_args_t *p_args)
IRQn_Type	mint_irq MINT interrupt IRQ number.
ptp_device_t	device PTP clock type.
ptp_state_t	state [2] PTP clock state.
ptp_delay_mechanism_t	delay [2] PTP delay correction mechanism.
ptp_frame_format_t	frame_format [2] PTP message frame format.

ptp_stca_mode_t	stca_mode STCA synchronous mode.
ptp_address_t	address [2] IP and MAC address.
uint32_t	open Flag to determine if the device is open.
uint8_t	eptpc_flag [2] Flag to check the EPTPC channel.
uint8_t	infabt_flag [2] Flag to check INFABT status.
void *	p_reg_gen Pointer to R_EPTPC_GEN_Type base register.
void *	p_reg_cfg Pointer to R_EPTPC_CFG_Type base register.
void *	p_reg [2] Pointer to R_EPTPC0_Type base register.
void const *	p_context Placeholder for user data.
void const *	p_extend Extension parameter for hardware specific settings.

Detailed Description

PTP instance control block. DO NOT INITIALIZE. Initialized in ptp_api_t::open().

Field Documentation

◆ p_callback

ssp_err_t(* ptp_instance_ctrl_t::p_callback) (ptp_callback_args_t *p_args)
Callback provided when a PTP message is received. NULL indicates no CPU interrupt.

The documentation for this struct was generated from the following file:

- r_ptp.h

5.1.5.45 PTPEDMAC

Renesas Synergy Software Package Reference » HAL Layer

DMA controller for PTP driver. More...

Data Structures

struct	ptpedmac_instance_ctrl_t

Functions

ssp_err_t	R_PTPEDMAC_Open (ptpedmac_ctrl_t *const p_api_ctrl, ptpedmac_cfg_t const *const p_cfg) Open the PTP host interface (PTPEDMAC), handles required initialization described in hardware manual. Implements ptpedmac_api_t::open. More...
ssp_err_t	R_PTPEDMAC_LinkProcess (ptpedmac_ctrl_t *const p_api_ctrl) Sets PTP host interface to transfer PTP messages. Implements ptpedmac_api_t::linkProcess. More...
ssp_err_t	R_PTPEDMAC_CheckLink (ptpedmac_ctrl_t *const p_api_ctrl) Checks PTP host interface communication link. Implements ptpedmac_api_t::checkLink. More...
ssp_err_t	R_PTPEDMAC_Read (ptpedmac_ctrl_t *const p_api_ctrl, uint32_t *p_channel, void *p_buffer, int32_t *p_num_received) Receive PTP message. Implements ptpedmac_api_t::read. More...

ssp_err_t	R_PTPEDMAC_Close (ptpedmac_ctrl_t *const p_api_ctrl) Disable PTP host interface. Implements ptpedmac_api_t::close. More...
ssp_err_t	R_PTPEDMAC_VersionGet (ssp_version_t *const p_version) This function returns the API version. Implements ptpedmac_api_t::versionGet. More...

Detailed Description

DMA controller for PTP driver.

Summary

This module implements the following interface: PTPEDMAC driver Interface.

Function Documentation

◆ R_PTPEDMAC_CheckLink()

ssp_err_t R_PTPEDMAC_CheckLink (ptpedmac_ctrl_t *const *p_api_ctrl*)

Checks PTP host interface communication link. Implements ptpedmac_api_t::checkLink.

Return values

SSP_SUCCESS	PTP host interface link status is verified successfully
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	PTPEDMAC driver is not opened.
SSP_ERR_NOT_ENABLED	PTP host interface is not enabled

Check the current status of PTP message transfer

◆ R_PTPEDMAC_Close()

ssp_err_t R_PTPEDMAC_Close (ptpedmac_ctrl_t *const *p_api_ctrl*)

Disable PTP host interface. Implements ptpedmac_api_t::close.

Return values

SSP_SUCCESS	PTP host interface is closed successfully
SSP_ERR_NOT_OPEN	PTPEDMAC driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

Clear PTPEDMAC interrupt

Set PTP Host interface transfer flag to disable

The device is now considered closed

◆ R_PTPEDMAC_LinkProcess()

ssp_err_t R_PTPEDMAC_LinkProcess (ptpedmac_ctrl_t *const *p_api_ctrl*)

Sets PTP host interface to transfer PTP messages. Implements ptpedmac_api_t::linkProcess.

Return values

SSP_SUCCESS	PTP host interface has linked successfully to transfer PTP messages
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	PTPEDMAC driver is not opened.

Initialize the receive descriptor

Initialize and configure PTPEDMAC

Set PTP Host interface transfer flag to enable

◆ R_PTPEDMAC_Open()

ssp_err_t R_PTPEDMAC_Open (ptpedmac_ctrl_t *const *p_api_ctrl*, ptpedmac_cfg_t const *const *p_cfg*)

Open the PTP host interface (PTPEDMAC), handles required initialization described in hardware manual. Implements ptpedmac_api_t::open.

Return values

SSP_SUCCESS	PTP host interface has opened successfully and initialization was successful.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Reset PTPEDMAC

Wait at least 64 cycles of PCLKA to reset the PTPEDMAC and EPTPC. PCLKA must be at least 12.5 MHz to use Ethernet, so wait at least 5.12 us.

Make sure the peripheral exists.

Initialize the channel state information.

If interrupt is registered in the vector table, disable interrupts, set priority, and store control block in the vector information so it can be accessed from the callback.

Enable PINT interrupt

Initialize frame transfer status flag

Mark driver as opened by initializing it to "DMAC" in its ASCII equivalent for this unit.

◆ R_PTPEDMAC_Read()

ssp_err_t R_PTPEDMAC_Read (ptpedmac_ctrl_t *const *p_api_ctrl*, uint32_t * *p_channel*, void * *p_buffer*, int32_t * *p_num_received*)

Receive PTP message. Implements ptpedmac_api_t::read.

Return values

SSP_SUCCESS	PTP message received successfully
SSP_ERR_TIMEOUT	No data received
SSP_ERR_NOT_OPEN	PTPEDMAC driver is not opened.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_ENABLED	PTP host interface is not enabled

Set the allocated buffer pointer for received data

◆ R_PTPEDMAC_VersionGet()

ssp_err_t R_PTPEDMAC_VersionGet (ssp_version_t *const *p_version*)

This function returns the API version. Implements ptpedmac_api_t::versionGet.

Parameters

[in]	p_version	Pointer to the API version block

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

ptpedmac_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » PTPEDMAC

#include <r_ptpedmac.h>

Public Member Functions

ptpedmac_descriptor_t BSP_ALIGN_VARIABLE_V2 (16) p_rx_descriptors [PTPEDMAC

_NUM_RX_DESCRIPTORS]	Pointer to receive descriptor aligned to 16 bytes.

Data Fields

ssp_err_t(*	p_callback)(ptpedmac_callback_args_t *p_args)
void const *	p_context Pointer to user interrupt context data.
IRQn_Type	pint_irq PINT interrupt IRQ number.
void *	p_reg Pointer to R_PTPEDMAC_Type base register.
uint32_t	open Flag to determine if the device is open.
ptpedmac_trans_t	transfer_flag Flag to determine if the PTP host interface is linked.
ptpedmac_ether_buffer_t	p_ptpedmac_buffer Pointer to Ethernet buffer.
ptpedmac_descriptor_t *	p_app_ptp_rx_desc Pointer to application descriptor.

Detailed Description

PTPEDMAC instance control block. DO NOT INITIALIZE. Initialized in ptpedmac_api_t::open().

Field Documentation

◆ p_callback

ssp_err_t(* ptpedmac_instance_ctrl_t::p_callback) (ptpedmac_callback_args_t *p_args)
Callback provided when a PTP message is received. NULL indicates no CPU interrupt.

The documentation for this struct was generated from the following file:

- r_ptpedmac.h

5.1.5.46 QSPI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Quad Serial Peripheral Interface (QSPI). More...

Data Structures

struct	qspi_instance_ctrl_t

Functions

ssp_err_t	R_QSPI_Open (qspi_ctrl_t *p_api_ctrl, qspi_cfg_t const *const p_cfg) Open the QSPI driver module. More...
ssp_err_t	R_QSPI_Close (qspi_ctrl_t *p_api_ctrl) Close the QSPI driver module. More...
ssp_err_t	R_QSPI_Read (qspi_ctrl_t *p_api_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count) Read data from the flash. More...
ssp_err_t	R_QSPI_PageProgram (qspi_ctrl_t *p_api_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count) Program a page of data to the flash. More...
ssp_err_t	R_QSPI_Erase (qspi_ctrl_t *p_api_ctrl, uint8_t *p_device_address, uint32_t byte_count) Erase a number of byte from the flash. More...

ssp_err_t	R_QSPI_InfoGet (qspi_ctrl_t *p_api_ctrl, qspi_info_t *const p_info) Returns the information about the flash. More...
ssp_err_t	R_QSPI_SectorErase (qspi_ctrl_t *p_api_ctrl, uint8_t *p_device_address) Erase a sector on the flash. More...
ssp_err_t	R_QSPI_StatusGet (qspi_ctrl_t *p_api_ctrl, bool *p_write_in_progress) Get the write or erase status of the flash. More...
ssp_err_t	R_QSPI_BankSelect (uint32_t bank) Select the bank to access. More...
ssp_err_t	R_QSPI_VersionGet (ssp_version_t *const p_version) Get the driver version based on compile time macros. More...

Detailed Description

Driver for the Quad Serial Peripheral Interface (QSPI).

This is a driver for the Quad-SPI module (QSPI) which is a memory controller for connecting a serial ROM (non-volatile memory such as a serial flash memory, serial EEPROM, or serial FeRAM) that has an SPI-compatible interface.

Summary

Extends Quad SPI Flash Interface.

Function Documentation

◆ R_QSPI_BankSelect()

ssp_err_t R_QSPI_BankSelect (uint32_t *bank*)

Select the bank to access.

A bank is a 64MB sliding access window into the flash memory space. This function sets the current bank.

Return values

SSP_SUCCESS	Bank successfully selected.

Return back to ROM access mode

◆ R_QSPI_Close()

ssp_err_t R_QSPI_Close (qspi_ctrl_t * *p_api_ctrl*)

Close the QSPI driver module.

Return the QSPI module back to ROM access mode.

Return values

SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check if the device is open

Re-enter XIP mode if it was running in this mode before entering opening the driver

Clearing the manufacturing_id, memory_type and memory_capacity

◆ R_QSPI_Erase()

ssp_err_t R_QSPI_Erase (qspi_ctrl_t * *p_api_ctrl*, uint8_t * *p_device_address*, uint32_t *byte_count*)

Erase a number of byte from the flash.

Return values

SSP_SUCCESS	The command to erase the flash was executed successfully.
SSP_ERR_UNSUPPORTED	The device address is invalid.
SSP_ERR_ASSERTION	p_ctrl or p_device_address is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid byte_count entered.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check if the device is open

Check whether the device address is valid

Get the information of underlying flash

If requested byte_count is supported by underlying flash, assign the value of size_index to cmd_index for searching the command

Send command to enable writing

Get the erase command

Send command to erase

If the command is not a chip erase command then send the start address

Send command to write data

Send command to disable writing

◆ R_QSPI_InfoGet()

ssp_err_t R_QSPI_InfoGet (qspi_ctrl_t * *p_api_ctrl*, qspi_info_t *const *p_info*)

Returns the information about the flash.

Return values

SSP_SUCCESS	Operation was successful.
SSP_ERR_ASSERTION	p_ctrl or p_info is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check if the device is open

Get the information of underlying flash

◆ R_QSPI_Open()

ssp_err_t R_QSPI_Open (qspi_ctrl_t * *p_api_ctrl*, qspi_cfg_t const *const *p_cfg*)

Open the QSPI driver module.

Open the QSPI module driver in direct communication mode for the purposes of reading and writing flash memory via SPI protocols.

Return values

SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_UNSUPPORTED	Driver not able to query the following information from the flash manufacturer id,memory capacity and memory type.
SSP_ERR_IN_USE	QSPI resource is locked.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Set the default bank to the first bank

Get the configuration of the quad SPI flash device and remember it for subsequent operations

If populated flash is 16MB & address mode configured as 4-byte,returns an unsupported error

A zero in the manufacturer_id mean the flash device is broken, misconfigured, or not populated

Mark driver as opened by initializing it to "RQSP" in its ASCII equivalent for this unit.

Exit XIP mode while the driver is open

◆ R_QSPI_PageProgram()

ssp_err_t R_QSPI_PageProgram (qspi_ctrl_t * *p_api_ctrl*, uint8_t * *p_device_address*, uint8_t * *p_memory_address*, uint32_t *byte_count*)

Program a page of data to the flash.

Return values

SSP_SUCCESS	The flash was programmed successfully.
SSP_ERR_UNSUPPORTED	The device address is invalid.
SSP_ERR_ASSERTION	p_ctrl, p_device_address or p_memory_address is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check whether the device address is valid

Send command to enable writing

If the peripheral is in extended SPI mode, and the configuration provided in the BSP allows for programming on multiple data lines, and a unique command is provided for the required mode, update the SPI protocol to send data on multiple lines.

Send command to write data

Write the address.

Write the data.

If the SPI protocol was modified in this function, restore it.

Send command to disable writing

◆ R_QSPI_Read()

ssp_err_t R_QSPI_Read (qspi_ctrl_t * *p_api_ctrl*, uint8_t * *p_device_address*, uint8_t * *p_memory_address*, uint32_t *byte_count*)

Read data from the flash.

Read a block of data from a particular address on the SPI flash device.

Return values

SSP_SUCCESS	The flash was programmed successfully.
SSP_ERR_UNSUPPORTED	The device address is invalid.
SSP_ERR_ASSERTION	p_ctrl,p_device_address or p_memory_address is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_TRANSFER_BUSY	Another serial communications transfer is in progress.

Check if the device is open

Check whether the device address is valid

Make sure no other communication is in progress.

◆ R_QSPI_SectorErase()

ssp_err_t R_QSPI_SectorErase (qspi_ctrl_t * *p_api_ctrl*, uint8_t * *p_device_address*)

Erase a sector on the flash.

Erase one sector on the SPI flash device. Any passed in address within the sector to be erased is acceptable.

Return values

SSP_SUCCESS	The command to erase the sector of flash was executed successfully.
SSP_ERR_UNSUPPORTED	The device address is invalid.
SSP_ERR_ASSERTION	p_ctrl or p_device_address is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check whether the device address is valid

Check if the device is open

Place the QSPI block into Direct Communication mode

Send command to enable writing

Close the SPI bus cycle

Send command to erase

Send command to write data

Send command to erase

Close the SPI bus cycle

Send command to disable writing

Close the SPI bus cycle

Return to ROM access mode

◆ R_QSPI_StatusGet()

ssp_err_t R_QSPI_StatusGet (qspi_ctrl_t * *p_api_ctrl*, bool * *p_write_in_progress*)

Get the write or erase status of the flash.

Return the write status of the flash. This is most useful for determining if erases are complete.

Return values

SSP_SUCCESS	The write status is correct.
SSP_ERR_ASSERTION	p_ctrl or p_write_in_progress is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

Check if the device is open

Place the QSPI block into Direct Communication mode

Get the write status from the device

Return to ROM access mode

◆ R_QSPI_VersionGet()

ssp_err_t R_QSPI_VersionGet (ssp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

qspi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » QSPI

#include <r_qspi.h>

Data Fields

R_QSPI_Type *	p_reg Pointer to QSPI base register.
uint32_t	max_eraseable_size

uint32_t	num_address_bytes Number of bytes used to represent the address.
uint32_t	spi_mode SPI mode - 0 = Extended, 1 = Dual, 2 = Quad.
uint32_t	page_size Number of bytes in a programmable page.
uint8_t	data_lines data lines - 0 = 1 line, 1 = 2 lines, 2 = 4 lines
uint8_t	manufacturer_id Manufacturer ID.
uint8_t	memory_type Memory type.
uint8_t	memory_capacity Memory capacity (in MByte)
bool	xip_mode 0 = run in read mode, 1 = run in XIP mode
uint32_t	total_size_bytes Total size of the flash in bytes.
uint32_t	open Flag to determine if the device is open.

Detailed Description

Instance control block. DO NOT INITIALIZE. Initialization occurs when qspi_api_t::open is called

Field Documentation

◆ max_eraseable_size

uint32_t qspi_instance_ctrl_t::max_eraseable_size
Largest eraseable sector size in kbytes. Used to determine buffer size for partial sector erases.

The documentation for this struct was generated from the following file:

- r_qspi.h

5.1.5.47 IIC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the I2C Bus Interface (IIC). More...

Data Structures

struct	riic_instance_ctrl_t
struct	riic_extended_cfg

Macros

#define	RIIC_OPEN (0x52494943ULL)
#define	RIIC_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_master_version)

Enumerations

enum	riic_timeout_mode_t { RIIC_TIMEOUT_MODE_LONG = 0, RIIC_TIMEOUT_MODE_SHORT = 1 }

Functions

ssp_err_t	R_RIIC_MasterVersionGet (ssp_version_t *const p_version) Gets version information and stores it in the provided version struct. More...
ssp_err_t	R_RIIC_MasterOpen (i2c_ctrl_t *const p_api_ctrl, i2c_cfg_t const *const p_cfg) Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual. More...

ssp_err_t	R_RIIC_MasterClose (i2c_ctrl_t *const p_api_ctrl) Closes the I2C device. May power down IIC peripheral. More...
ssp_err_t	R_RIIC_MasterRead (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) Performs a read from the I2C device. More...
ssp_err_t	R_RIIC_MasterWrite (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) Performs a write to the I2C device. More...
ssp_err_t	R_RIIC_MasterReset (i2c_ctrl_t *const p_api_ctrl) Aborts any in-progress transfer and forces the IIC peripheral into a ready state. More...
ssp_err_t	R_RIIC_MasterSlaveAddressSet (i2c_ctrl_t *const p_api_ctrl, uint16_t const slave_address, i2c_addr_mode_t const addr_mode) Sets address and addressing mode of the slave device. More...

Variables

i2c_api_master_t const	g_i2c_master_on_riic

Detailed Description

Driver for the I2C Bus Interface (IIC).

This module supports the Renesas Inter-Integrated Circuit (IIC) peripheral. It implements the following interfaces:

- I2C Interface r_i2c_api.h

Macro Definition Documentation

◆ RIIC_ERROR_RETURN

#define RIIC_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_master_version)
Macro for error logger.

◆ RIIC_OPEN

#define RIIC_OPEN (0x52494943ULL)
"RIIC" in ASCII, used to determine if channel is open.

Enumeration Type Documentation

◆ riic_timeout_mode_t

enum riic_timeout_mode_t	
I2C Timeout mode parameter definition	
Enumerator	
RIIC_TIMEOUT_MODE_LONG	Timeout Detection Time Select: Long Mode -> TMOS = 0.
RIIC_TIMEOUT_MODE_SHORT	Timeout Detection Time Select: Short Mode -> TMOS = 1.

Function Documentation

◆ R_RIIC_MasterClose()

ssp_err_t R_RIIC_MasterClose (i2c_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. May power down IIC peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

SSP_SUCCESS	Device closed without issue.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.

Check if the device is even open, return an error if not

Abort an in-progress transfer with this device only

Close the DTC transfer interfaces if configured

Disable the interrupt sources for I2C peripheral

◆ R_RIIC_MasterOpen()

ssp_err_t R_RIIC_MasterOpen (i2c_ctrl_t *const *p_api_ctrl*, i2c_cfg_t const *const *p_cfg*)

Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual.

This function will reconfigure the clock settings of the peripheral when a device with a lower rate than previously configured is opened.

Return values

SSP_SUCCESS	Requested clock rate was set exactly.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_cfg is NULL or clock rate is greater than 1MHz. or the extended parameter is NULL
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_INVALID_ARGUMENT	If fast mode plus is configured and the channel does not support it
SSP_ERR_INVALID_RATE	The requested rate cannot be set.

Returns

See Common Error Codes for other possible return codes. This function calls
- fmi_api_t::productFeatureGet
- g_cgc_on_cgc.systemClockFreqGet

If rate is configured as Fast mode plus, check whether the channel supports it

set valid interrupts with user provided priority

Open the hardware in master mode

Open the RIIC DTC transfer interface if enabled

Initialize control block

◆ R_RIIC_MasterRead()

ssp_err_t R_RIIC_MasterRead (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue, if no callback was provided, the process was kicked off.
SSP_ERR_ASSERTION	p_api_ctrl, p_dest or bytes is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_HW_LOCKED	Driver busy doing RIIC operation
SSP_ERR_ABORTED	The transfer failed.

Check if the device is even open, return an error if not

Attempt to acquire lock for this transfer operation. Prevents re-entrance conflict.

Record the new information about this transfer

Kickoff the read operation as a master

◆ R_RIIC_MasterReset()

ssp_err_t R_RIIC_MasterReset (i2c_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the IIC peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

SSP_SUCCESS	Channel was reset without issue.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	A transfer was aborted while resetting the hardware.

Check if the device is even open, return an error if not

Abort any on-going transfer on the channel

◆ R_RIIC_MasterSlaveAddressSet()

ssp_err_t R_RIIC_MasterSlaveAddressSet (i2c_ctrl_t *const *p_api_ctrl*, uint16_t const *slave_address*, i2c_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

SSP_SUCCESS	Address of the slave is set correctly.
SSP_ERR_ASSERTION	Pointer to control structure is NULL.
SSP_ERR_HW_LOCKED	Driver busy doing RIIC operation.
SSP_ERR_NOT_OPEN	Device was not even opened.

Check if the device is open, return an error if not

Attempt to acquire lock for configuring the slave address. Prevents re-entrance conflict.

Return failure if there is already a transfer in progress

Sets the address of the slave device

Sets the mode of addressing

◆ R_RIIC_MasterVersionGet()

ssp_err_t R_RIIC_MasterVersionGet (ssp_version_t *const *p_version*)

Gets version information and stores it in the provided version struct.

Return values

SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	p_version is NULL.

◆ R_RIIC_MasterWrite()

ssp_err_t R_RIIC_MasterWrite (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue, if no callback was provided, the process was kicked off.
SSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_HW_LOCKED	Driver busy doing RIIC operation
SSP_ERR_ABORTED	The transfer failed.

Check if the device is even open, return an error if not

Attempt to acquire lock for this transfer operation. Prevents re-entrance conflict.

Record the new information about this transfer

Kickoff the write operation as a master

Variable Documentation

◆ g_i2c_master_on_riic

i2c_api_master_t const g_i2c_master_on_riic

```
=
{
    .open            = R_RIIC_MasterOpen,
    .close           = R_RIIC_MasterClose,
    .read            = R_RIIC_MasterRead,
    .write           = R_RIIC_MasterWrite,
    .reset           = R_RIIC_MasterReset,
    .versionGet      = R_RIIC_MasterVersionGet,
    .slaveAddressSet = R_RIIC_MasterSlaveAddressSet
}
```

RIIC Implementation of I2C device master interface

riic_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » IIC

#include <r_riic.h>

Data Fields

i2c_cfg_t	info Information describing I2C device.
uint32_t	open Flag to determine if the device is open.
void *	p_reg Base register for this channel.
IRQn_Type	rxi_irq Receive IRQ number.

IRQn_Type	txi_irq Transmit IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
IRQn_Type	eri_irq Error IRQ number.
uint8_t *	p_buff
uint32_t	total
uint32_t	remain
uint32_t	loaded
uint8_t	addr_low
uint8_t	addr_high
uint8_t	addr_total
uint8_t	addr_remain
uint8_t	addr_loaded
volatile bool	read
volatile bool	restart
volatile bool	err
volatile bool	restarted
volatile bool	dummy_read_completed
volatile bool	activation_on_rxi
volatile bool	activation_on_txi
volatile bool	address_restarted
volatile bsp_lock_t	resource_lock_tx_rx

riic_timeout_mode_t	timeout_mode
i2c_hw_err_event_t	actual_hwErr_event

Detailed Description

I2C control structure. DO NOT INITIALIZE.

Field Documentation

◆ activation_on_rxi

volatile bool riic_instance_ctrl_t::activation_on_rxi

Tracks whether the transfer is activated on RXI interrupt

◆ activation_on_txi

volatile bool riic_instance_ctrl_t::activation_on_txi

Tracks whether the transfer is activated on TXI interrupt

◆ actual_hwErr_event

i2c_hw_err_event_t riic_instance_ctrl_t::actual_hwErr_event

Holds error event value obtained through hardware

◆ addr_high

uint8_t riic_instance_ctrl_t::addr_high

Holds the first address byte to issue in 10-bit mode

◆ addr_loaded

uint8_t riic_instance_ctrl_t::addr_loaded

Tracks the number of address bytes written to the register

◆ addr_low

uint8_t riic_instance_ctrl_t::addr_low

Holds the last address byte to issue

◆ addr_remain

uint8_t riic_instance_ctrl_t::addr_remain
Tracks the remaining address bytes to transfer

◆ addr_total

uint8_t riic_instance_ctrl_t::addr_total
Holds the total number of address bytes to transfer

◆ address_restarted

volatile bool riic_instance_ctrl_t::address_restarted
Tracks whether the restart condition is send on 10 bit read

◆ dummy_read_completed

volatile bool riic_instance_ctrl_t::dummy_read_completed
Tracks whether the dummy read is performed

◆ err

volatile bool riic_instance_ctrl_t::err
Tracks whether or not an error occurred during processing

◆ loaded

uint32_t riic_instance_ctrl_t::loaded
Tracks the number of data bytes written to the register

◆ p_buff

uint8_t* riic_instance_ctrl_t::p_buff
Holds the data associated with the transfer

◆ read

volatile bool riic_instance_ctrl_t::read
Holds the direction of the data byte transfer

◆ remain

uint32_t riic_instance_ctrl_t::remain
Tracks the remaining data bytes to transfer

◆ resource_lock_tx_rx

volatile bsp_lock_t riic_instance_ctrl_t::resource_lock_tx_rx
Resource lock for transmission/reception

◆ restart

volatile bool riic_instance_ctrl_t::restart
Holds whether or not the restart should be issued when done

◆ restarted

volatile bool riic_instance_ctrl_t::restarted
Tracks whether or not a restart was issued during the previous transfer

◆ timeout_mode

riic_timeout_mode_t riic_instance_ctrl_t::timeout_mode
Holds the timeout mode value. i.e short mode or long mode

◆ total

uint32_t riic_instance_ctrl_t::total
Holds the total number of data bytes to transfer

The documentation for this struct was generated from the following file:

- r_riic.h

riic_extended_cfg Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » IIC

#include <r_riic.h>

Data Fields

riic_timeout_mode_t	timeout_mode Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1.

Detailed Description

R_IIC extended configuration

The documentation for this struct was generated from the following file:

- r_riic.h

5.1.5.48 IIC Slave

Renesas Synergy Software Package Reference » HAL Layer

Driver for the I2C Bus Slave Interface (IIC Slave). More...

Data Structures

struct	riic_slave_instance_ctrl_t

Macros

#define	RIIC_SLAVE_ERROR_RETURN(a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_slave_version)
#define	RIIC_SLAVE_OPEN (0x49324353ULL)

Functions

ssp_err_t	R_RIIC_SlaveVersionGet (ssp_version_t *const p_version) Gets version information and stores it in the provided version struct.

	More...
ssp_err_t	R_RIIC_SlaveOpen (i2c_ctrl_t *const p_api_ctrl, i2c_cfg_t const *const p_cfg) Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual. More...
ssp_err_t	R_RIIC_SlaveClose (i2c_ctrl_t *const p_api_ctrl) Closes the I2C device. Power down IIC peripheral. More...
ssp_err_t	R_RIIC_MasterWriteSlaveRead (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) Performs a read from the I2C Master device. More...
ssp_err_t	R_RIIC_MasterReadSlaveWrite (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) Performs a write to the I2C Master device. More...

Variables

i2c_api_slave_t const	g_i2c_slave_on_riic

Detailed Description

Driver for the I2C Bus Slave Interface (IIC Slave).

This module supports the Renesas Inter-Integrated Circuit (IIC) peripheral. It implements the following interfaces:

- I2C Interface r_i2c_api.h

Macro Definition Documentation

◆ RIIC_SLAVE_ERROR_RETURN

#define RIIC_SLAVE_ERROR_RETURN (a, err) SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_slave_version)
Macro for error logger.

◆ RIIC_SLAVE_OPEN

#define RIIC_SLAVE_OPEN (0x49324353ULL)
"I2CS" in ASCII, used to determine if channel is open.

Function Documentation

◆ R_RIIC_MasterReadSlaveWrite()

ssp_err_t R_RIIC_MasterReadSlaveWrite (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue; if no callback was provided, the process was kicked off
SSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_NOT_OPEN	device is not open.
SSP_ERR_ABORTED	If transaction encounter an error.

Check if the device is open, return an error if not

Return an error if transfer is in progress

Record the new information about this transfer

Start the write operation as a slave

◆ R_RIIC_MasterWriteSlaveRead()

ssp_err_t R_RIIC_MasterWriteSlaveRead (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue; if no callback was provided, the process was kicked off
SSP_ERR_ASSERTION	p_api_ctrl, bytes or p_dest is NULL.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_NOT_OPEN	device is not open.
SSP_ERR_ABORTED	If transaction encounter an error.

Check if the device is open, return an error if not

Return an error if transfer is in progress.

Record the new information about this transfer

Start the read operation as a slave

◆ R_RIIC_SlaveClose()

ssp_err_t R_RIIC_SlaveClose (i2c_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down IIC peripheral.

Return values

SSP_SUCCESS	Device closed without issue.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.

Check if the device is even open, return an error if not

De-configure everything.

◆ R_RIIC_SlaveOpen()

ssp_err_t R_RIIC_SlaveOpen (i2c_ctrl_t *const *p_api_ctrl*, i2c_cfg_t const *const *p_cfg*)

Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual.

Return values

SSP_SUCCESS	Opened identical configuration of already open instance.
SSP_ERR_ASSERTION	p_api_ctrl or p_cfg is NULL.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.
SSP_ERR_INVALID_ARGUMENT	If fast mode plus is configured and the channel does not support it

Returns

See Common Error Codes for other possible return codes. This function calls
- fmi_api_t::productFeatureGet
- g_cgc_on_cgc.systemClockFreqGet

If rate is configured as Fast mode plus, check whether the channel supports it

Attempt to acquire hardware lock

Open the hardware in slave mode

Clear all interrupt bits

Enable both TXI and RXI interrupt sources

Set ACK as slave is now ready to serve requests from master

Enable all RIIC interrupts in NVIC, that need to be serviced

Release hardware lock on failure

◆ R_RIIC_SlaveVersionGet()

ssp_err_t R_RIIC_SlaveVersionGet (ssp_version_t *const *p_version*)

Gets version information and stores it in the provided version struct.

Return values

SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	p_version is NULL.

Variable Documentation

◆ g_i2c_slave_on_riic

i2c_api_slave_t const g_i2c_slave_on_riic

```
=
{
    .open                       = R_RIIC_SlaveOpen,
    .close                       = R_RIIC_SlaveClose,
    .masterWriteSlaveRead       = R_RIIC_MasterWriteSlaveRead,
    .masterReadSlaveWrite       = R_RIIC_MasterReadSlaveWrite,
    .versionGet                 = R_RIIC_SlaveVersionGet
}
```

RIIC Implementation of I2C device slave interface

riic_slave_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » IIC Slave

#include <r_riic_slave.h>

Data Fields

i2c_cfg_t	info Information describing I2C device.
uint32_t	open Flag to determine if the device is open.
void *	p_reg Base register for this channel.
IRQn_Type	rxi_irq Receive IRQ number.
IRQn_Type	txi_irq

	Transmit IRQ number.
IRQn_Type	eri_irq Error IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
uint8_t *	p_buff
uint32_t	total
uint32_t	remain
uint32_t	loaded
uint32_t	transaction_count
volatile bool	notify_request
volatile bool	read
volatile bool	err
volatile bool	slave_busy
volatile bool	start_interrupt_enabled

Detailed Description

I2C control structure. DO NOT INITIALIZE.

Field Documentation

◆ err

volatile bool riic_slave_instance_ctrl_t::err
Tracks whether or not an error occurred during processing

◆ loaded

uint32_t riic_slave_instance_ctrl_t::loaded
Tracks the number of data bytes written to the register

◆ notify_request

volatile bool riic_slave_instance_ctrl_t::notify_request
Track whether the master request is notified to the application

◆ p_buff

uint8_t* riic_slave_instance_ctrl_t::p_buff
Holds the data associated with the transfer

◆ read

volatile bool riic_slave_instance_ctrl_t::read
Holds the direction of the data byte transfer

◆ remain

uint32_t riic_slave_instance_ctrl_t::remain
Tracks the remaining data bytes to transfer

◆ slave_busy

volatile bool riic_slave_instance_ctrl_t::slave_busy
Tracks if the slave is busy performing a transaction

◆ start_interrupt_enabled

volatile bool riic_slave_instance_ctrl_t::start_interrupt_enabled
<< Tracks whether a dummy read is issued on the first RX < Tracks whether the start interrupt is enabled

◆ total

uint32_t riic_slave_instance_ctrl_t::total
Holds the total number of data bytes to transfer

◆ transaction_count

uint32_t riic_slave_instance_ctrl_t::transaction_count
Tracks the actual number of transactions

The documentation for this struct was generated from the following file:

- r_riic_slave.h

5.1.5.49 SPI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Serial Peripheral Interface (SPI). More...

Data Structures

struct	rspi_ssl_polarity_t
struct	rspi_loopback_t
struct	rspi_mosi_idle_t
struct	rspi_parity_t
struct	rspi_clock_delay_t
struct	rspi_ssl_negation_delay_t
struct	rspi_access_delay_t
struct	spi_on_rspi_cfg_t
struct	rspi_instance_ctrl_t

Enumerations

enum	rspi_spcmd_bit_length_t { RSPI_SPCMD_BIT_LENGTH_8 = (0x7), RSPI_SPCMD_BIT_LENGTH_16 = (0xF), RSPI_SPCMD_BIT_LENGTH_32 = (0x3) }
enum	rspi_operation_t { RSPI_OPERATION_SPI, RSPI_OPERATION_CLK_SYN }
enum	rspi_communication_t { RSPI_COMMUNICATION_FULL_DUPLEX, RSPI_COMMUNICATION_TRANSMIT_ONLY }
enum	rspi_sslp_t { RSPI_SSLP_LOW, RSPI_SSLP_HIGH }
enum	rspi_loopback1_t { RSPI_LOOPBACK1_NORMAL_DATA, RSPI_LOOPBACK1_INVERTED_DATA }
enum	rspi_loopback2_t { RSPI_LOOPBACK2_NORMAL_DATA, RSPI_LOOPBACK2_NOT_INVERTED_DATA }
enum	rspi_mosi_idle_fixed_val_t { RSPI_MOSI_IDLE_FIXED_VAL_LOW, RSPI_MOSI_IDLE_FIXED_VAL_HIGH }
enum	rspi_mosi_idle_val_fixing_t { RSPI_MOSI_IDLE_VAL_FIXING_ENABLE, RSPI_MOSI_IDLE_VAL_FIXING_DISABLE }
enum	rspi_parity_state_t { RSPI_PARITY_STATE_DISABLE, RSPI_PARITY_STATE_ENABLE }
enum	rspi_byte_swap_t { RSPI_BYTE_SWAP_DISABLE, RSPI_BYTE_SWAP_ENABLE }
enum	rspi_parity_mode_t { RSPI_PARITY_MODE_ODD, RSPI_PARITY_MODE_EVEN }
enum	rspi_ssl_select_t { RSPI_SSL_SELECT_SSL0, RSPI_SSL_SELECT_SSL1, RSPI_SSL_SELECT_SSL2, RSPI_SSL_SELECT_SSL3 }
enum	rspi_ssl_level_keep_t { RSPI_SSL_LEVEL_KEEP_NOT, RSPI_SSL_LEVEL_KEEP }
enum	rspi_clock_delay_count_t { RSPI_CLOCK_DELAY_COUNT_1, RSPI_CLOCK_DELAY_COUNT_2, RSPI_CLOCK_DELAY_COUNT_3, RSPI_CLOCK_DELAY_COUNT_4, RSPI_CLOCK_DELAY_COUNT_5, RSPI_CLOCK_DELAY_COUNT_6, RSPI_CLOCK_DELAY_COUNT_7, RSPI_CLOCK_DELAY_COUNT_8 }
enum	rspi_clock_delay_state_t { RSPI_CLOCK_DELAY_STATE_DISABLE, RSPI_CLOCK_DELAY_STATE_ENABLE }
enum	rspi_ssl_negation_delay_count_t { RSPI_SSL_NEGATION_DELAY_1, RSPI_SSL_NEGATION_DELAY_2,

	RSPI_SSL_NEGATION_DELAY_3, RSPI_SSL_NEGATION_DELAY_4, RSPI_SSL_NEGATION_DELAY_5, RSPI_SSL_NEGATION_DELAY_6, RSPI_SSL_NEGATION_DELAY_7, RSPI_SSL_NEGATION_DELAY_8 }
enum	rspi_ssl_negation_delay_state_t { RSPI_SSL_NEGATION_DELAY_DISABLE, RSPI_SSL_NEGATION_DELAY_ENABLE }
enum	rspi_next_access_delay_count_t { RSPI_NEXT_ACCESS_DELAY_COUNT_1, RSPI_NEXT_ACCESS_DELAY_COUNT_2, RSPI_NEXT_ACCESS_DELAY_COUNT_3, RSPI_NEXT_ACCESS_DELAY_COUNT_4, RSPI_NEXT_ACCESS_DELAY_COUNT_5, RSPI_NEXT_ACCESS_DELAY_COUNT_6, RSPI_NEXT_ACCESS_DELAY_COUNT_7, RSPI_NEXT_ACCESS_DELAY_COUNT_8 }
enum	rspi_next_access_delay_state_t { RSPI_NEXT_ACCESS_DELAY_STATE_DISABLE, RSPI_NEXT_ACCESS_DELAY_STATE_ENABLE }
enum	rspi_spcmd_br_div_t { RSPI_SPCMD_BR_DIV_1 = (0x0), RSPI_SPCMD_BR_DIV_2 = (0x1), RSPI_SPCMD_BR_DIV_4 = (0x2), RSPI_SPCMD_BR_DIV_8 = (0x3) }
enum	rspi_spcmd_assert_ssl_t { RSPI_SPCMD_ASSERT_SSL0 = (0x0), RSPI_SPCMD_ASSERT_SSL1 = (0x1), RSPI_SPCMD_ASSERT_SSL2 = (0x2), RSPI_SPCMD_ASSERT_SSL3 = (0x3) }

Functions

ssp_err_t	R_RSPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) This functions initializes a channel for SPI communication mode. More...
ssp_err_t	R_RSPI_Read (spi_ctrl_t *const p_api_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width) This function receives data from a SPI device. More...
ssp_err_t	R_RSPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) This function transmits data to a SPI device using the TX Only Communications Operation Mode. More...
ssp_err_t	R_RSPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src,

	void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width) This function simultaneously transmits data to a SPI device while receiving data from a SPI device (full duplex). More...
ssp_err_t	R_RSPI_Close (spi_ctrl_t *const p_api_ctrl) This function manages the closing of a channel by the following task. More...
ssp_err_t	R_RSPI_VersionGet (ssp_version_t *p_version) This function gets the version information of the underlying driver. More...

Detailed Description

Driver for the Serial Peripheral Interface (SPI).

This module supports SPI serial communication for the SPI module. The SPI Interface is defined in r_spi_api.h

Enumeration Type Documentation

◆ rspi_byte_swap_t

enum rspi_byte_swap_t	
SPDCR2 (RSPI Data Control Register 2) – Byte swapping operation enable/disable	
Enumerator	
RSPI_BYTE_SWAP_DISABLE	Disable Byte swap
RSPI_BYTE_SWAP_ENABLE	Enable Byte swap

◆ rspi_clock_delay_count_t

enum rspi_clock_delay_count_t	
SPCKD (RSPI Clock Delay) Register – Clock Delay Count select	
Enumerator	
RSPI_CLOCK_DELAY_COUNT_1	Set RSPCK Clock delay to 1 RSPCK
RSPI_CLOCK_DELAY_COUNT_2	Set RSPCK Clock delay to 2 RSPCK
RSPI_CLOCK_DELAY_COUNT_3	Set RSPCK Clock delay to 3 RSPCK
RSPI_CLOCK_DELAY_COUNT_4	Set RSPCK Clock delay to 4 RSPCK
RSPI_CLOCK_DELAY_COUNT_5	Set RSPCK Clock delay to 5 RSPCK
RSPI_CLOCK_DELAY_COUNT_6	Set RSPCK Clock delay to 6 RSPCK
RSPI_CLOCK_DELAY_COUNT_7	Set RSPCK Clock delay to 7 RSPCK
RSPI_CLOCK_DELAY_COUNT_8	Set RSPCK Clock delay to 8 RSPCK

◆ rspi_clock_delay_state_t

enum rspi_clock_delay_state_t	
SPCMD (RSPI Command) Register – RSPCK Delay Enable/Disable select – SCKDEN	
Enumerator	
RSPI_CLOCK_DELAY_STATE_DISABLE	RSPCK delay=1 RSPCK
RSPI_CLOCK_DELAY_STATE_ENABLE	RSPCK delay=SPCKD register setting

◆ rspi_communication_t

enum rspi_communication_t	
SPCR (RSPI Control register) – TXMD (communication operating mode) select	
Enumerator	
RSPI_COMMUNICATION_FULL_DUPLEX	Full-Duplex synchronous serial communication
RSPI_COMMUNICATION_TRANSMIT_ONLY	Transit only serial communication

◆ rspi_loopback1_t

enum rspi_loopback1_t	
SPPCR (RSPI Pin Control Register) – Loopback1 select	
Enumerator	
RSPI_LOOPBACK1_NORMAL_DATA	Loopback1 normal mode
RSPI_LOOPBACK1_INVERTED_DATA	Loopback1 with inverted data

◆ rspi_loopback2_t

enum rspi_loopback2_t	
SPPCR (RSPI Pin Control Register) – Loopback2 select	
Enumerator	
RSPI_LOOPBACK2_NORMAL_DATA	Loopback2 normal mode
RSPI_LOOPBACK2_NOT_INVERTED_DATA	Loopback2 with not inverted data

◆ rspi_mosi_idle_fixed_val_t

enum rspi_mosi_idle_fixed_val_t	
SPPCR (RSPI Pin Control Register) – MOIFV select	
Enumerator	
RSPI_MOSI_IDLE_FIXED_VAL_LOW	MOSIn level low during MOSI idling
RSPI_MOSI_IDLE_FIXED_VAL_HIGH	MOSIn level high during MOSI idling

◆ rspi_mosi_idle_val_fixing_t

enum rspi_mosi_idle_val_fixing_t	
SPPCR (RSPI Pin Control Register) – MOIFE (MOSI idle value fixing) select	
Enumerator	
RSPI_MOSI_IDLE_VAL_FIXING_ENABLE	MOSI output value=final data from previous transfer
RSPI_MOSI_IDLE_VAL_FIXING_DISABLE	MOSI output value=value set in MOIFV bit

◆ rspi_next_access_delay_count_t

enum rspi_next_access_delay_count_t	
SPND (RSPI Next-Access Delay) Register – Next Access Delay Count select	
Enumerator	
RSPI_NEXT_ACCESS_DELAY_COUNT_1	Set next access delay to 1 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_2	Set next access delay to 2 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_3	Set next access delay to 3 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_4	Set next access delay to 4 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_5	Set next access delay to 5 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_6	Set next access delay to 6 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_7	Set next access delay to 7 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_8	Set next access delay to 8 RSPCK+2PCLK

◆ rspi_next_access_delay_state_t

enum rspi_next_access_delay_state_t	
SPCMD (RSPI Command) Register – Next Access Delay select – SPNDEN	
Enumerator	
RSPI_NEXT_ACCESS_DELAY_STATE_DISABLE	Next access delay=1 RSPCK + 2 PCLK
RSPI_NEXT_ACCESS_DELAY_STATE_ENABLE	Next access delay=SPND register setting

◆ rspi_operation_t

enum rspi_operation_t	
SPCR (RSPI Control register) – SPMS (RSPI mode) select	
Enumerator	
RSPI_OPERATION_SPI	SPI operation (4-wire method)
RSPI_OPERATION_CLK_SYN	Clock Synchronous operation (3-wire method)

◆ rspi_parity_mode_t

enum rspi_parity_mode_t	
SPCR2 (RSPI Control Register 2) – Parity select	
Enumerator	
RSPI_PARITY_MODE_ODD	Select even parity
RSPI_PARITY_MODE_EVEN	Select odd parity

◆ rspi_parity_state_t

enum rspi_parity_state_t	
SPCR2 (RSPI Control Register 2) – Parity Enable select	
Enumerator	
RSPI_PARITY_STATE_DISABLE	Disable parity
RSPI_PARITY_STATE_ENABLE	Enable parity

◆ rspi_spcmd_assert_ssl_t

enum rspi_spcmd_assert_ssl_t	
Slave select to be asserted during transfer operation.	
Enumerator	
RSPI_SPCMD_ASSERT_SSL0	Select SSL0
RSPI_SPCMD_ASSERT_SSL1	Select SSL1
RSPI_SPCMD_ASSERT_SSL2	Select SSL2
RSPI_SPCMD_ASSERT_SSL3	Select SSL3

◆ rspi_spcmd_bit_length_t

enum rspi_spcmd_bit_length_t	
Frame data length	
Enumerator	
RSPI_SPCMD_BIT_LENGTH_8	0100 to 0111 = 8 bits data length
RSPI_SPCMD_BIT_LENGTH_16	1111 = 16 bits data length
RSPI_SPCMD_BIT_LENGTH_32	0011 = 32 bits data length

◆ rspi_spcmd_br_div_t

enum rspi_spcmd_br_div_t	
Clock base rate division	
Enumerator	
RSPI_SPCMD_BR_DIV_1	Select the base bit rate
RSPI_SPCMD_BR_DIV_2	Select the base bit rate divided by 2
RSPI_SPCMD_BR_DIV_4	Select the base bit rate divided by 4
RSPI_SPCMD_BR_DIV_8	Select the base bit rate divided by 8

◆ rspi_ssl_level_keep_t

enum rspi_ssl_level_keep_t	
SPCMD (RSPI Command) Register – SSL Signal Level Keeping select	
Enumerator	
RSPI_SSL_LEVEL_KEEP_NOT	Negates all SSL signals upon transfer completion
RSPI_SSL_LEVEL_KEEP	Keeps the SSL level upon transfer completion

◆ rspi_ssl_negation_delay_count_t

enum rspi_ssl_negation_delay_count_t	
SSLND (RSPI Slave Select Negation Delay) Register – Slave Select Negation Delay Count select	
Enumerator	
RSPI_SSL_NEGATION_DELAY_1	Set SSL negation delay to 1 RSPCK
RSPI_SSL_NEGATION_DELAY_2	Set SSL negation delay to 2 RSPCK
RSPI_SSL_NEGATION_DELAY_3	Set SSL negation delay to 3 RSPCK
RSPI_SSL_NEGATION_DELAY_4	Set SSL negation delay to 4 RSPCK
RSPI_SSL_NEGATION_DELAY_5	Set SSL negation delay to 5 RSPCK
RSPI_SSL_NEGATION_DELAY_6	Set SSL negation delay to 6 RSPCK
RSPI_SSL_NEGATION_DELAY_7	Set SSL negation delay to 7 RSPCK
RSPI_SSL_NEGATION_DELAY_8	Set SSL negation delay to 8 RSPCK

◆ rspi_ssl_negation_delay_state_t

enum rspi_ssl_negation_delay_state_t	
SPCMD (RSPI Command) Register – SSL Negation Delay select – SLNDEN	
Enumerator	
RSPI_SSL_NEGATION_DELAY_DISABLE	SSL negation delay=1 RSPCK
RSPI_SSL_NEGATION_DELAY_ENABLE	SSL negation delay=SSLND register setting

◆ rspi_ssl_select_t

enum rspi_ssl_select_t	
SPCMD (RSPI Command) Register – SSL Signal Assertion select	
Enumerator	
RSPI_SSL_SELECT_SSL0	Select SSL0 as slave
RSPI_SSL_SELECT_SSL1	Select SSL1 as slave
RSPI_SSL_SELECT_SSL2	Select SSL2 as slave
RSPI_SSL_SELECT_SSL3	Select SSL3 as slave

◆ rspi_sslp_t

enum rspi_sslp_t	
Definition for SSLP (RSPI Slave Select Polarity register) select	
Enumerator	
RSPI_SSLP_LOW	SSLP signal polarity active low
RSPI_SSLP_HIGH	SSLP signal polarity active high

Function Documentation

◆ R_RSPI_Close()

ssp_err_t R_RSPI_Close (spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task.

Implements spi_api_t::close Disables SPI operations by disabling the SPI bus. Power off the channel. Disables all the associated interrupts. Update channel status.

Return values

SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	A required pointer argument is NULL.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

Note

This function is reentrant.

◆ R_RSPI_Open()

ssp_err_t R_RSPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

This functions initializes a channel for SPI communication mode.

Implements spi_api_t::open This function performs the following tasks: Performs parameter checking and processes error conditions. Applies power to the SPI channel. Disables interrupts. Initializes the associated registers with some default value and the user-configurable options. Provides the channel control for use with other API functions. Updates user-configurable file if necessary.

Return values

SSP_SUCCESS	Channel initialized successfully.
SSP_ERR_ASSERTION	NULL pointer to following parameters p_ctrl, p_cfg, p_cfg::p_transfer_rx::p_api, p_cfg::p_transfer_rx::p_ctrl, p_cfg::p_transfer_rx::p_cfg, p_cfg::p_transfer_rx::p_cfg::p_info. or failed to set the baud rate,
SSP_ERR_INVALID_ARGUMENT	An element of the r_spi_cfg_t structure contains an invalid value. The parameters is out of range. Both transfer modules need to be present or absent.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open
- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Note

This function is reentrant.

◆ R_RSPI_Read()

ssp_err_t R_RSPI_Read (spi_ctrl_t *const *p_api_ctrl*, void const * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function receives data from a SPI device.

Implements spi_api_t::read The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission with dummy data via transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination. Complete data reception via receive buffer full interrupt and transmitting dummy data.

Return values

SSP_SUCCESS	Read operation successfully completed.
SSP_ERR_ASSERTION	NULL pointer to control or destination parameters or transfer length is zero.
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

Note

This function is reentrant.

◆ R_RSPI_VersionGet()

ssp_err_t R_RSPI_VersionGet (ssp_version_t * *p_version*)

This function gets the version information of the underlying driver.

Implements spi_api_t::versionget

Return values

void	
SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Note

This function is reentrant.

◆ R_RSPI_Write()

ssp_err_t R_RSPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function transmits data to a SPI device using the TX Only Communications Operation Mode.

Implements spi_api_t::write The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission with dummy data via transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and do nothing with the received data. Complete data transmission via receive buffer full interrupt.

Return values

SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	NULL pointer to control or source parameters or transfer length is zero.
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

Note

This function is reentrant.

◆ R_RSPI_WriteRead()

ssp_err_t R_RSPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void const * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function simultaneously transmits data to a SPI device while receiving data from a SPI device (full duplex).

Implements spi_api_t::writeread The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission using transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination. Complete data transmission and reception via receive buffer full interrupt.

Return values

SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	NULL pointer to control, source or destination parameters or transfer length is zero.
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

Note

This function is reentrant.

rspi_ssl_polarity_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

SSLP (RSPI Slave Select Polarity register) – SSLnP select

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_loopback_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

SPPCR (RSPI Pin Control Register) – Loopback select

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_mosi_idle_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

SPPCR (RSPI Pin Control Register) – MOIFV (mosi idle value) select

---

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_parity_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

SPCR2 (RSPI Control Register 2) – Parity select

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_clock_delay_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

Select RSPI Clock Delay Register (SPCKD) and SPCMD (RSPI Command) Register-Clock Delay state(SCKDEN)

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_ssl_negation_delay_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

Select SSL Negation Delay(SSLND) and SPCMD Register-SSL negation Delay state(SLNDEN)

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_access_delay_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Detailed Description

Select Next Access Delay(SPND) and SPCMD Register-Next Access Delay state(SPNDEN)

The documentation for this struct was generated from the following file:

- r_rspi.h

spi_on_rspi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Data Fields

rspi_operation_t	rspi_clksyn
rspi_communication_t	rspi_comm
rspi_ssl_polarity_t	ssl_polarity
rspi_loopback_t	loopback
rspi_mosi_idle_t	mosi_idle
rspi_parity_t	parity
rspi_ssl_select_t	ssl_select
rspi_ssl_level_keep_t	ssl_level_keep
rspi_clock_delay_t	clock_delay
rspi_ssl_negation_delay_t	ssl_neg_delay
rspi_access_delay_t	access_delay
rspi_byte_swap_t	byte_swap

Detailed Description

Extended SPI interface configuration

Field Documentation

◆ access_delay

rspi_access_delay_t spi_on_rspi_cfg_t::access_delay
Select next access delay from 0 to 7

◆ byte_swap

rspi_byte_swap_t spi_on_rspi_cfg_t::byte_swap
Feature for byte swapping

◆ clock_delay

rspi_clock_delay_t spi_on_rspi_cfg_t::clock_delay
Select clock delay from 0 to 7

◆ loopback

rspi_loopback_t spi_on_rspi_cfg_t::loopback
Select loopback1 and loopback2

◆ mosi_idle

rspi_mosi_idle_t spi_on_rspi_cfg_t::mosi_idle
Select mosi idle fixed value and selection

◆ parity

rspi_parity_t spi_on_rspi_cfg_t::parity
Select parity and enable/disable parity

◆ rspi_clksyn

rspi_operation_t spi_on_rspi_cfg_t::rspi_clksyn
Select spi or clock syn mode operation

◆ rspi_comm

rspi_communication_t spi_on_rspi_cfg_t::rspi_comm
Select full-duplex or transmit-only communication

◆ ssl_level_keep

rspi_ssl_level_keep_t spi_on_rspi_cfg_t::ssl_level_keep
Select SSL level after transfer completion;0-negate;1-keeps

◆ ssl_neg_delay

rspi_ssl_negation_delay_t spi_on_rspi_cfg_t::ssl_neg_delay
Select Slave elect negation delay from 0 to 7

◆ ssl_polarity

rspi_ssl_polarity_t spi_on_rspi_cfg_t::ssl_polarity
Select SSLn signal polarity

◆ ssl_select

rspi_ssl_select_t spi_on_rspi_cfg_t::ssl_select
Select which slave to use;0-SSL0;1-SSL1;2-SSL2;3-SSL3

The documentation for this struct was generated from the following file:

- r_rspi.h

rspi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SPI

#include <r_rspi.h>

Data Fields

uint8_t	channel Channel number to be used.
uint8_t	current_slave Number of the currently assigned slave.
uint32_t	channel_opened Internal flag to indicate the peripheral was initialized.
transfer_instance_t const *	p_transfer_tx To use SPI DTC/DMA write transfer.
transfer_instance_t const *	p_transfer_rx To use SPI DTC/DMA read transfer.
void(*	p_callback)(spi_callback_args_t *p_args) Pointer to user callback function.
void const *	p_context Pointer to the higher level device context.
void *	p_reg Base register for this channel.
IRQn_Type	rxi_irq Receive IRQ number.
IRQn_Type	txi_irq Transmit IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
IRQn_Type	eri_irq

	Error IRQ number.

Detailed Description

SPI instance control block. DO NOT INITIALIZE.

The documentation for this struct was generated from the following file:

- r_rspi.h

5.1.5.50 RTC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Realtime Clock (RTC). More...

Data Structures

struct	rtc_instance_ctrl_t

Enumerations

enum	rtc_count_mode_t { RTC_CALENDAR_MODE = 0, RTC_BINARY_MODE = 1 }

Functions

ssp_err_t	R_RTC_Open (rtc_ctrl_t *const p_api_ctrl, rtc_cfg_t const *const p_cfg) Open the RTC driver. More...
ssp_err_t	R_RTC_Close (rtc_ctrl_t *const p_api_ctrl) Close the RTC driver. More...
ssp_err_t	R_RTC_Configure (rtc_ctrl_t *const p_api_ctrl, void *const p_extend) Configure the RTC driver. More...
ssp_err_t	R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_api_ctrl, rtc_time_t *p_time, bool clock_start) Set the calendar time. More...

ssp_err_t	R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_api_ctrl, rtc_time_t *p_time) Get the calendar time. (Should not be called from critical section as it may return incorrect time) More...
ssp_err_t	R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_api_ctrl, rtc_alarm_time_t *p_alarm, bool interrupt_enable_flag) Set the calendar alarm time. More...
ssp_err_t	R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_api_ctrl, rtc_alarm_time_t *p_alarm) Get the calendar alarm time. More...
ssp_err_t	R_RTC_CalendarCounterStart (rtc_ctrl_t *const p_api_ctrl) Start the calendar counter. More...
ssp_err_t	R_RTC_CalendarCounterStop (rtc_ctrl_t *const p_api_ctrl) Stop the calendar counter. More...
ssp_err_t	R_RTC_IrqEnable (rtc_ctrl_t *const p_api_ctrl, rtc_event_t event) Enable the alarm interrupt. More...
ssp_err_t	R_RTC_IrqDisable (rtc_ctrl_t *const p_api_ctrl, rtc_event_t event) Disable the alarm interrupt. More...
ssp_err_t	R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_api_ctrl, rtc_periodic_irq_select_t rate) Set the periodic interrupt rate. More...
ssp_err_t	R_RTC_InfoGet (rtc_ctrl_t *p_api_ctrl, rtc_info_t *p_rtc_info) This function returns information about the driver clock source. More...
ssp_err_t	R_RTC_ErrorAdjustmentModeSet (rtc_ctrl_t *p_api_ctrl, rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode) This function sets time error adjustment mode. More...

ssp_err_t	R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *p_api_ctrl, rtc_error_adjustment_cfg_t *p_error_adjustment_config) This function sets time error adjustment. More...
ssp_err_t	R_RTC_VersionGet (ssp_version_t *p_version) Get driver version based on compile time macros. More...

Detailed Description

Driver for the Realtime Clock (RTC).

This module supports the Real Time Clock (RTC). It implements the following interfaces:

- RTC Interface

Enumeration Type Documentation

◆ rtc_count_mode_t

enum rtc_count_mode_t	
Counting mode	
Enumerator	
RTC_CALENDAR_MODE	Calender count mode.
RTC_BINARY_MODE	Binary count mode.

Function Documentation

◆ R_RTC_CalendarAlarmGet()

ssp_err_t R_RTC_CalendarAlarmGet (rtc_ctrl_t *const *p_api_ctrl*, rtc_alarm_time_t * *p_alarm*)

Get the calendar alarm time.

Implements rtc_api_t::calendarAlarmGet

Return values

SSP_SUCCESS	Calendar alarm time get operation was successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_alarm or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

Get the alarm time

◆ R_RTC_CalendarAlarmSet()

ssp_err_t R_RTC_CalendarAlarmSet (rtc_ctrl_t *const *p_api_ctrl*, rtc_alarm_time_t * *p_alarm*, bool *interrupt_enable_flag*)

Set the calendar alarm time.

Implements rtc_api_t::calendarAlarmSet.

Precondition

The calendar counter must be running before the alarm can be set.

Return values

SSP_SUCCESS	Calendar alarm time set operation was successful.
SSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_alarm or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

Disable the ICU alarm interrupt request

Set alarm time

◆ R_RTC_CalendarCounterStart()

ssp_err_t R_RTC_CalendarCounterStart (rtc_ctrl_t *const *p_api_ctrl*)

Start the calendar counter.

Implements rtc_api_t::calendarCounterStart.

Return values

SSP_SUCCESS	Calendar counter started.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	Start bit not set.

Set the start bit.

◆ R_RTC_CalendarCounterStop()

ssp_err_t R_RTC_CalendarCounterStop (rtc_ctrl_t *const *p_api_ctrl*)

Stop the calendar counter.

Implements rtc_api_t::calendarCounterStop.

Return values

SSP_SUCCESS	Calendar counter stopped.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	Start bit not cleared.

Clear the start bit.

◆ R_RTC_CalendarTimeGet()

ssp_err_t R_RTC_CalendarTimeGet (rtc_ctrl_t *const *p_api_ctrl*, rtc_time_t * *p_time*)

Get the calendar time. (Should not be called from critical section as it may return incorrect time)

Implements rtc_api_t::calendarTimeGet

Return values

SSP_SUCCESS	Calendar time get operation was successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_time or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	IRQ enable operation timed out.

Read all the time registers, if a carry irq occurred in-between read again

This flag will be set to 'true' in the carry ISR

Restore the state of carry IRQ.

◆ R_RTC_CalendarTimeSet()

ssp_err_t R_RTC_CalendarTimeSet (rtc_ctrl_t *const *p_api_ctrl*, rtc_time_t * *p_time*, bool *clock_start*)

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

Return values

SSP_SUCCESS	Calendar time set operation was successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_time or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
SSP_ERR_TIMEOUT	Software reset status check failed.

◆ R_RTC_Close()

ssp_err_t R_RTC_Close (rtc_ctrl_t *const *p_api_ctrl*)

Close the RTC driver.

Implements rtc_api_t::close

Return values

SSP_SUCCESS	De-Initialization was successful and RTC driver closed.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for close.

Disable the periodic interrupt, alarm interrupt, carry interrupts, and disable there interrupt priority and vector info.

◆ R_RTC_Configure()

ssp_err_t R_RTC_Configure (rtc_ctrl_t *const *p_api_ctrl*, void *const *p_extend*)

Configure the RTC driver.

Implements rtc_api_t::configure

Return values

SSP_SUCCESS	RTC was successful configured.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	Status check for counter mode or reset timed out

Parameter checking

p_extend is currently not used, reserved for future use

Set the clock source of the RTC block according to the UM

◆ R_RTC_ErrorAdjustmentModeSet()

ssp_err_t R_RTC_ErrorAdjustmentModeSet (rtc_ctrl_t * *p_api_ctrl*, rtc_error_adjustment_mode_cfg_t * *p_error_adjustment_mode*)

This function sets time error adjustment mode.

Implements rtc_api_t::errorAdjustmentModeSet

Return values

SSP_SUCCESS	Time error adjustment mode set successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or error_adjustment_mode pointer.
SSP_ERR_NOT_OPEN	Driver not open for operation.
SSP_ERR_UNSUPPORTED	The clock source is not SubClock.
SSP_ERR_INVALID_ARGUMENT	Invalid error adjustment period.
SSP_ERR_TIMEOUT	Time error adjustment get query timed out.

◆ R_RTC_ErrorAdjustmentSet()

ssp_err_t R_RTC_ErrorAdjustmentSet (rtc_ctrl_t * *p_api_ctrl*, rtc_error_adjustment_cfg_t * *p_error_adjustment_config*)

This function sets time error adjustment.

Implements rtc_api_t::errorAdjustmentSet

Return values

SSP_SUCCESS	Time error adjustment successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_error_adjustment_config pointer.
SSP_ERR_NOT_OPEN	Driver not open for operation.
SSP_ERR_UNSUPPORTED	The clock source is not SubClock.
SSP_ERR_INVALID_ARGUMENT	Invalid error adjustment value.
SSP_ERR_TIMEOUT	Time error adjustment get query timed out.

◆ R_RTC_InfoGet()

ssp_err_t R_RTC_InfoGet (rtc_ctrl_t * *p_api_ctrl*, rtc_info_t * *p_rtc_info*)

This function returns information about the driver clock source.

Implements rtc_api_t::infoGet

Return values

SSP_SUCCESS	Get information Successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_rtc_info or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_IrqDisable()

ssp_err_t R_RTC_IrqDisable (rtc_ctrl_t *const *p_api_ctrl*, rtc_event_t *event*)

Disable the alarm interrupt.

Implements rtc_api_t::interruptDisable.

Return values

SSP_SUCCESS	Alarm interrupt disabled.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid
SSP_ERR_INVALID_ARGUMENT	Invalid IRQ event
SSP_ERR_TIMEOUT	IRQ disable operation timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_IrqEnable()

ssp_err_t R_RTC_IrqEnable (rtc_ctrl_t *const *p_api_ctrl*, rtc_event_t *event*)

Enable the alarm interrupt.

Implements rtc_api_t::interruptEnable.

Return values

SSP_SUCCESS	Alarm interrupt enabled.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid.
SSP_ERR_INVALID_ARGUMENT	Invalid IRQ event.
SSP_ERR_TIMEOUT	IRQ enable operation timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_Open()

ssp_err_t R_RTC_Open (rtc_ctrl_t *const *p_api_ctrl*, rtc_cfg_t const *const *p_cfg*)

Open the RTC driver.

Implements rtc_api_t::open.

Opens and configures the RTC driver module. Configuration includes clock source, and interrupt callback function. If the sub-clock oscillator is the clock source it is started in this function.

Return values

SSP_SUCCESS	Initialization was successful and RTC has started.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_cfg pointer.
SSP_ERR_HW_LOCKED	Hardware in use
SSP_ERR_TIMEOUT	Status check for counter mode or reset timed out

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Mark driver as open by initializing it to "RTC" in its ASCII equivalent.

◆ R_RTC_PeriodicIrqRateSet()

ssp_err_t R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const *p_api_ctrl*, rtc_periodic_irq_select_t *rate*)

Set the periodic interrupt rate.

Implements rtc_api_t::periodicInterruptRateSet.

Return values

SSP_SUCCESS	The periodic interrupt rate was successfully set.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_TIMEOUT	Periodic interrupt rate get query timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_VersionGet()

ssp_err_t R_RTC_VersionGet (ssp_version_t * *p_version*)

Get driver version based on compile time macros.

Implements rtc_api_t::versionGet

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

rtc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » RTC

#include <r_rtc.h>

Data Fields

void *	p_reg Pointer to register base address.
void(*	p_callback)(rtc_callback_args_t *cb_data)

	Called from the ISR.
void const *	p_context Passed to the callback.
uint32_t	open Whether or not driver is open.
IRQn_Type	alarm_irq Alarm IRQ number.
IRQn_Type	periodic_irq Periodic IRQ number.
IRQn_Type	carry_irq Carry IRQ number.
rtc_clock_source_t	clock_source Clock source for the RTC block.
bool	suppress_carry_event_callback carry event callback will be suppressed if set
volatile bool	carry_isr_triggered Was the carry isr triggered.

Detailed Description

Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called

---

The documentation for this struct was generated from the following file:

- r_rtc.h

5.1.5.51 Simple I2C on SCI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Simple IIC on SCI. More...

Data Structures

struct	sci_i2c_instance_ctrl_t
struct	sci_i2c_extended_cfg

Functions

ssp_err_t	R_SCI_SIIC_MasterVersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
ssp_err_t	R_SCI_SIIC_MasterOpen (i2c_ctrl_t *const p_api_ctrl, i2c_cfg_t const *const p_cfg) Opens the I2C device. Power on I2C peripheral and perform initialization described in hardware manual. More...
ssp_err_t	R_SCI_SIIC_MasterClose (i2c_ctrl_t *const p_api_ctrl) Closes the I2C device. Power down I2C peripheral. More...
ssp_err_t	R_SCI_SIIC_MasterRead (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) Performs a read from the I2C device. More...
ssp_err_t	R_SCI_SIIC_MasterWrite (i2c_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) Performs a write to the I2C device. More...
ssp_err_t	R_SCI_SIIC_MasterReset (i2c_ctrl_t *const p_api_ctrl) Aborts any in-progress transfer and forces the I2C peripheral into a ready state. More...
ssp_err_t	R_SCI_SIIC_MasterSlaveAddressSet (i2c_ctrl_t *const p_api_ctrl, uint16_t const slave, i2c_addr_mode_t const addr_mode) Sets address and addressing mode of the slave device. More...

Detailed Description

Driver for the Simple IIC on SCI.

This module supports the SCI in I2C mode. It implements the following interfaces:

- I2C Interface r_i2c_api.h

Function Documentation

◆ R_SCI_SIIC_MasterClose()

ssp_err_t R_SCI_SIIC_MasterClose (i2c_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

SSP_SUCCESS	Device closed without issue.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_ABORTED	Device was closed while a transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

Abort an in-progress transfer with this device only

The device is now considered closed

De-configure everything.

◆ R_SCI_SIIC_MasterOpen()

ssp_err_t R_SCI_SIIC_MasterOpen (i2c_ctrl_t *const *p_api_ctrl*, i2c_cfg_t const *const *p_cfg*)

Opens the I2C device. Power on I2C peripheral and perform initialization described in hardware manual.

This function will reconfigure the clock settings of the peripheral when a device with a lower rate than previously configured is opened.

Return values

SSP_SUCCESS	Requested baud rate was valid.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL or if clock rate greater than 400KHz.
SSP_ERR_INVALID_RATE	The requested rate cannot be set.
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet

Disable, clear and configure interrupts

Open the hardware in master mode

◆ R_SCI_SIIC_MasterRead()

ssp_err_t R_SCI_SIIC_MasterRead (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue.
SSP_ERR_ASSERTION	The parameter p_ctrl, p_dest is NULL, bytes is 0.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

Record the new information about this transfer

Handle the different addressing modes

Kickoff the read operation as a master

◆ R_SCI_SIIC_MasterReset()

ssp_err_t R_SCI_SIIC_MasterReset (i2c_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

SSP_SUCCESS	Channel was reset without issue.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_ABORTED	A transfer was aborted while resetting the hardware.
SSP_ERR_NOT_OPEN	Device was not even opened.

Abort any transfer happening on the channel

◆ R_SCI_SIIC_MasterSlaveAddressSet()

ssp_err_t R_SCI_SIIC_MasterSlaveAddressSet (i2c_ctrl_t *const *p_api_ctrl*, uint16_t const *slave*, i2c_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

SSP_SUCCESS	Address of the slave is set correctly.
SSP_ERR_ASSERTION	p_ctrl or address is NULL.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

Sets the address of the slave device

Sets the mode of addressing

◆ R_SCI_SIIC_MasterVersionGet()

ssp_err_t R_SCI_SIIC_MasterVersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Verify parameter is valid

◆ R_SCI_SIIC_MasterWrite()

ssp_err_t R_SCI_SIIC_MasterWrite (i2c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

SSP_SUCCESS	Function executed without issue.
SSP_ERR_ASSERTION	p_ctrl, p_src is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

Record the new information about this transfer

Handle the different addressing modes

Kickoff the write operation as a master

sci_i2c_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Simple I2C on SCI

#include <r_sci_i2c.h>

Data Fields

i2c_cfg_t	info Information describing I2C device.
uint32_t	open Flag to determine if the device is open.
void *	p_reg

	Base register for this channel.
transfer_instance_t const *	p_transfer_tx DTC instance for I2C transmit.Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx DTC instance for I2C receive. Set to NULL if unused.
IRQn_Type	rxi_irq Receive IRQ number.
IRQn_Type	txi_irq Transmit IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
uint8_t *	p_buff
uint32_t	total
uint32_t	remain
uint32_t	loaded
uint8_t	addr_low
uint8_t	addr_high
uint8_t	addr_total
uint8_t	addr_remain
uint8_t	addr_loaded
volatile bool	read
volatile bool	restart
volatile bool	err

volatile bool	restarted
volatile bool	transaction_completed
volatile bool	activation_on_rxi
volatile bool	activation_on_txi

Detailed Description

I2C control structure. DO NOT INITIALIZE.

Field Documentation

◆ activation_on_rxi

volatile bool sci_i2c_instance_ctrl_t::activation_on_rxi

<< Tracks whether a dummy read is issued on the first RX < Tracks whether the transfer is activated on RXI interrupt

◆ activation_on_txi

volatile bool sci_i2c_instance_ctrl_t::activation_on_txi

< Tracks whether the transfer is activated on TXI interrupt

◆ addr_high

uint8_t sci_i2c_instance_ctrl_t::addr_high

Holds the first address byte to issue in 10-bit mode

◆ addr_loaded

uint8_t sci_i2c_instance_ctrl_t::addr_loaded

Tracks the number of address bytes written to the register

◆ addr_low

uint8_t sci_i2c_instance_ctrl_t::addr_low

Holds the last address byte to issue

◆ addr_remain

uint8_t sci_i2c_instance_ctrl_t::addr_remain
Tracks the remaining address bytes to transfer

◆ addr_total

uint8_t sci_i2c_instance_ctrl_t::addr_total
Holds the total number of address bytes to transfer

◆ err

volatile bool sci_i2c_instance_ctrl_t::err
Tracks whether or not an error occurred during processing

◆ loaded

uint32_t sci_i2c_instance_ctrl_t::loaded
Tracks the number of data bytes written to the register

◆ p_buff

uint8_t* sci_i2c_instance_ctrl_t::p_buff
Holds the data associated with the transfer

◆ p_transfer_tx

transfer_instance_t const* sci_i2c_instance_ctrl_t::p_transfer_tx
DTC instance for I2C transmit.Set to NULL if unused. DTC/DMA support

◆ read

volatile bool sci_i2c_instance_ctrl_t::read
Holds the direction of the data byte transfer

◆ remain

uint32_t sci_i2c_instance_ctrl_t::remain
Tracks the remaining data bytes to transfer

◆ restart

volatile bool sci_i2c_instance_ctrl_t::restart
Holds whether or not the restart should be issued when done

◆ restarted

volatile bool sci_i2c_instance_ctrl_t::restarted
Tracks whether or not a restart was issued during the previous transfer

◆ total

uint32_t sci_i2c_instance_ctrl_t::total
Holds the total number of data bytes to transfer

◆ transaction_completed

volatile bool sci_i2c_instance_ctrl_t::transaction_completed
Tracks if the transaction started earlier was completed

The documentation for this struct was generated from the following file:

- r_sci_i2c.h

sci_i2c_extended_cfg Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Simple I2C on SCI

#include <r_sci_i2c.h>

Data Fields

bool	bitrate_modulation

Detailed Description

SCI I2C extended configuration

Field Documentation

◆ bitrate_modulation

bool sci_i2c_extended_cfg::bitrate_modulation
Bitrate Modulation Function enable or disable

The documentation for this struct was generated from the following file:

- r_sci_i2c.h

5.1.5.52 Simple SPI on SCI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Simple SPI on SCI. More...

Data Structures

struct	sci_spi_instance_ctrl_t
struct	sci_spi_extended_cfg

Functions

ssp_err_t	R_SCI_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) Initialize a channel for SPI communication mode. Implements spi_api_t::open. This function performs the following tasks: Performs parameter checking and processes error conditions. Applies power to the SPI channel. Disables interrupts. Initializes the associated registers with default value and the user-configurable options. Provides the channel handle for use with other API functions. Updates user-configurable file if necessary. More...
ssp_err_t	R_SCI_SPI_Read (spi_ctrl_t *const p_api_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width) Receive data from an SPI device. Implements spi_api_t::read. The

	function performs the following tasks: More...
ssp_err_t	R_SCI_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) Transmit data to a SPI device. Implements spi_api_t::write. More...
ssp_err_t	R_SCI_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width) Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead. The function performs the following tasks: More...
ssp_err_t	R_SCI_SPI_Close (spi_ctrl_t *const p_api_ctrl) Handle the closing of a channel by the following task. Implements spi_api_t::close Power off the channel. Disables all the associated interrupts. Update channel status. More...
ssp_err_t	R_SCI_SPI_VersionGet (ssp_version_t *p_version) Get the version information of the underlying driver. Implements spi_api_t::versionGet. More...

Variables

const spi_api_t	g_spi_on_sci

Detailed Description

Driver for the Simple SPI on SCI.

This module supports simple SPI serial communication using the microcontroller's SCI peripheral. The Interface is defined in r_spi_api.h. This module implements SPI Interface.

Function Documentation

◆ R_SCI_SPI_Close()

ssp_err_t R_SCI_SPI_Close (spi_ctrl_t *const *p_api_ctrl*)

Handle the closing of a channel by the following task. Implements spi_api_t::close Power off the channel. Disables all the associated interrupts. Update channel status.

Return values

SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

Note

This function is reentrant.

Check to see if the channel is currently initialized.

Turn off power.

Close transfer block.

Release lock for this channel.

◆ R_SCI_SPI_Open()

ssp_err_t R_SCI_SPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

Initialize a channel for SPI communication mode. Implements spi_api_t::open. This function performs the following tasks: Performs parameter checking and processes error conditions. Applies power to the SPI channel. Disables interrupts. Initializes the associated registers with default value and the user-configurable options. Provides the channel handle for use with other API functions. Updates user-configurable file if necessary.

Return values

SSP_SUCCESS	Channel initialized successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL • Pointer p_cfg is NULL
SSP_ERR_IN_USE	Channel currently in operation; Close channel first.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Note

This function is reentrant.
The bit-rate argument in p_cfg ranges from 2500 to 7.5m for Simple SPI at PCLK=120 MHz. For RSPI, BRDV is fixed at 0 to get the maximum bit rate. The range is 10.0 mbps to 30.0 mbps at PCLK=120.0 MHz.

Attempt to acquire lock for this SCI SPI channel. Prevents re-entrancy conflict.

set valid interrupts with user provided priority.

Turn on power.

Don't use FIFO mode – set FCRL_b.FM = 0.

Select SPI mode - set SCMR.SMIF=0, SIMR1.IICM=0, SMR.CM=1, SPMR.SSE=1.

Set baud rate in SCI channel for the SPI channel.

Open the SCI SPI transfer interface if available.

Peripheral Initialized.

Set control block for SCI channel to SPI mode operation.

◆ R_SCI_SPI_Read()

ssp_err_t R_SCI_SPI_Read (spi_ctrl_t *const *p_api_ctrl*, void const * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Receive data from an SPI device. Implements spi_api_t::read. The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Disable Interrupts.
- Set-up data bit width per user request.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission with dummy data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Return values

SSP_SUCCESS	Read operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Bit width is not 8 bits • Length is equal to 0 • Pointer to destination is NULL
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

This function is reentrant.

Configure module to receive data from a SPI device.

◆ R_SCI_SPI_VersionGet()

ssp_err_t R_SCI_SPI_VersionGet (ssp_version_t * *p_version*)

Get the version information of the underlying driver. Implements spi_api_t::versionGet.

Return values

SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Note

This function is reentrant.

◆ R_SCI_SPI_Write()

ssp_err_t R_SCI_SPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Transmit data to a SPI device. Implements spi_api_t::write.

- The function performs the following tasks:
- Performs parameter checking and processes error conditions.
- Disable Interrupts.
- Setup data bit width per user request.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt occurs and do nothing with the received data.
- Complete data transmission via receive buffer full interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Return values

SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

This function is reentrant.

Configure module to transmit data to a SPI device.

◆ R_SCI_SPI_WriteRead()

ssp_err_t R_SCI_SPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void const * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead. The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Disable Interrupts.
- Setup data bit width per user request.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data transmission and reception via receive buffer full interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Return values

SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Pointer to destination is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

This function is reentrant.

Configure module for full duplex operation.

Variable Documentation

◆ g_spi_on_sci

const spi_api_t g_spi_on_sci
Filled in Interface API structure for this Instance.

sci_spi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Simple SPI on SCI

#include <r_sci_spi.h>

Data Fields

uint8_t	channel Channel number to be used.
uint32_t	channel_opened Internal flag to indicate the peripheral was initialized.
transfer_instance_t const *	p_transfer_tx To use SPI DTC/DMA write transfer.
transfer_instance_t const *	p_transfer_rx To use SPI DTC/DMA read transfer.
void(*	p_callback)(spi_callback_args_t *p_args) Pointer to user callback function.
void const *	p_context Pointer to the higher level device context.
void *	p_reg Base register for this channel.
IRQn_Type	rxi_irq

	Receive IRQ number.
IRQn_Type	txi_irq Transmit IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
IRQn_Type	eri_irq Error IRQ number.
bsp_lock_t	resource_lock_tx_rx

Detailed Description

SPI instance control block. DO NOT INITIALIZE.

Field Documentation

◆ resource_lock_tx_rx

bsp_lock_t sci_spi_instance_ctrl_t::resource_lock_tx_rx
Resource lock for transmission/reception

The documentation for this struct was generated from the following file:

- r_sci_spi.h

sci_spi_extended_cfg Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Simple SPI on SCI

#include <r_sci_spi.h>

Data Fields

bool	bitrate_modulation

Detailed Description

SCI SPI extended configuration

Field Documentation

◆ bitrate_modulation

bool sci_spi_extended_cfg::bitrate_modulation
Bitrate Modulation Function enable or disable

The documentation for this struct was generated from the following file:

- r_sci_spi.h

5.1.5.53 UART on SCI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the UART on SCI. More...

Data Structures

struct	sci_uart_instance_ctrl_t
struct	uart_on_sci_cfg_t

Enumerations

enum	sci_clk_src_t { SCI_CLK_SRC_INT, SCI_CLK_SRC_EXT, SCI_CLK_SRC_EXT8X, SCI_CLK_SRC_EXT16X }
enum	sci_uart_rx_fifo_trigger_t { SCI_UART_RX_FIFO_TRIGGER_1 = 0x1, SCI_UART_RX_FIFO_TRIGGER_MAX = 0xF }

Functions

ssp_err_t	R_SCI_UartOpen (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg)
ssp_err_t	R_SCI_UartClose (uart_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SCI_UartRead (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_dest, uint32_t const bytes)
ssp_err_t	R_SCI_UartWrite (uart_ctrl_t *const p_api_ctrl, uint8_t const *const

	p_src, uint32_t const bytes)
ssp_err_t	R_SCI_UartBaudSet (uart_ctrl_t *const p_api_ctrl, uint32_t const baudrate)
ssp_err_t	R_SCI_UartInfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info)
ssp_err_t	R_SCI_UartVersionGet (ssp_version_t *p_version)
ssp_err_t	R_SCI_UartAbort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort)

Detailed Description

Driver for the UART on SCI.

Summary

This module supports the UART on SCI. It implements the UART interface and drives SCI as a full-duplex UART communication port. This module can drive all SCI channels as UART ports.

Extends UART Interface.

Note

This module can use either the 16-stage hardware FIFO or a DTC transfer implementation to write multiple bytes.

Enumeration Type Documentation

◆ sci_clk_src_t

enum sci_clk_src_t	
Enumeration for SCI clock source	
Enumerator	
SCI_CLK_SRC_INT	Use internal clock for baud generation.
SCI_CLK_SRC_EXT	Use external clock for baud generation.
SCI_CLK_SRC_EXT8X	Use external clock 8x baud rate.
SCI_CLK_SRC_EXT16X	Use external clock 16x baud rate.

◆ sci_uart_rx_fifo_trigger_t

enum sci_uart_rx_fifo_trigger_t	
Receive FIFO trigger configuration.	
Enumerator	
SCI_UART_RX_FIFO_TRIGGER_1	Callback after each byte is received without buffering.
SCI_UART_RX_FIFO_TRIGGER_MAX	Callback when FIFO is full or after 15 bit times with no data (fewer interrupts)

Function Documentation

◆ R_SCI_UartAbort()

ssp_err_t R_SCI_UartAbort (uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort*)

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR.

Return values

SSP_SUCCESS	UART transaction aborted successfully.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened.
SSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

◆ R_SCI_UartBaudSet()

ssp_err_t R_SCI_UartBaudSet (uart_ctrl_t *const *p_api_ctrl*, uint32_t const *baudrate*)

Updates the baud rate.

Warning

This terminates any in-progress transmission.

Return values

SSP_SUCCESS	Baud rate was successfully changed.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL or the UART is not configured to use the internal clock.
SSP_ERR_INVALID_ARGUMENT	Illegal baud rate value is specified.
SSP_ERR_NOT_OPEN	The control block has not been opened

Calculate new baud rate register settings.

Disables transmitter and receiver. This terminates any in-progress transmission.

Apply new baud rate register settings.

Enable receiver.

If the channel has no FIFO, enable transmitter.

◆ R_SCI_UartClose()

ssp_err_t R_SCI_UartClose (uart_ctrl_t *const *p_api_ctrl*)

Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power and releases hardware lock.

Return values

SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened

Mark the channel not open so other APIs cannot use it.

Disable interrupts, receiver, and transmitter.

If reception is enabled at build time, disable reception irqs.

If transmission is enabled at build time, disable transmission irqs.

Disable baud clock output.

Close the lower level transfer instances.

Clear control block parameters.

Remove power to the channel.

Unlock the SCI channel.

◆ R_SCI_UartInfoGet()

ssp_err_t R_SCI_UartInfoGet (uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info*)

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time.

Return values

SSP_SUCCESS	Information stored in provided p_info.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened

Store number of bytes that can be read at a time.

Store number of bytes that can be written at a time.

◆ R_SCI_UartOpen()

ssp_err_t R_SCI_UartOpen (uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg*)

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function.

Return values

SSP_SUCCESS	Channel opened successfully.
SSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure.
SSP_ERR_IN_USE	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- cgc_api_t::systemClockFreqGet
- transfer_api_t::open

Make sure this channel exists.

Reserve the hardware lock.

Determine if this channel has a FIFO.

Calculate the baud rate register settings.

Configure the interrupts.

Configure the transfer interface for transmission and reception if provided.

Configuration of driver enable pin for rs485 communication mode.

Enable the SCI channel and reset the registers to their initial state.

Set the default level of the TX pin to 1.

Set the baud rate registers.

Set the UART configuration settings provided in uart_cfg_t and uart_on_sci_cfg_t.

If reception is enabled at build time, enable reception.

◆ R_SCI_UartRead()

ssp_err_t R_SCI_UartRead (uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_dest*, uint32_t const *bytes*)

Receives user specified number of bytes into destination buffer pointer.

Return values

SSP_SUCCESS	Data reception successfully ends.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
SSP_ERR_INVALID_ARGUMENT	Destination address or data size is not valid for 9-bit mode.
SSP_ERR_NOT_OPEN	The control block has not been opened
SSP_ERR_IN_USE	A previous read operation is still in progress.
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_RX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

This API is only valid when SCI_UART_CFG_RX_ENABLE is enabled. If 9-bit data length is specified at R_SCI_UartOpen call, p_dest must be aligned 16-bit boundary.

Configure transfer instance to receive the requested number of bytes if transfer is used for reception.

◆ R_SCI_UartVersionGet()

ssp_err_t R_SCI_UartVersionGet (ssp_version_t * *p_version*)

Provides API and code version in the user provided pointer.

Parameters

[in]	p_version	Version number set here

Return values

SSP_SUCCESS	Version information stored in provided p_version.
SSP_ERR_ASSERTION	p_version is NULL.

◆ R_SCI_UartWrite()

ssp_err_t R_SCI_UartWrite (uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*)

Transmits user specified number of bytes from the source buffer pointer.

Return values

SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
SSP_ERR_INVALID_ARGUMENT	Source address or data size is not valid for 9-bit mode.
SSP_ERR_NOT_OPEN	The control block has not been opened
SSP_ERR_IN_USE	A UART transmission is in progress
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_TX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

This API is only valid when SCI_UART_CFG_TX_ENABLE is enabled. If 9-bit data length is specified at R_SCI_UartOpen call, p_src must be aligned on a 16-bit boundary.

Set the Driver Enable pin in RS485 half duplex mode to enable transmission.

Transmit interrupts must be disabled to start with.

Save data to transmit to the control block. It will be transmitted in the TXI ISR.

If a transfer instance is used for transmission, reset the transfer instance to transmit the requested data.

Clear the Driver Enable pin in RS485 half duplex mode to enable reception in error conditions

Trigger a TXI interrupt. This triggers the transfer instance or a TXI interrupt if the transfer instance is not used.

sci_uart_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » UART on SCI

#include <r_sci_uart.h>

Data Fields

uint8_t	channel Channel number.
uint8_t	fifo_depth FIFO depth of the UART channel.
uint8_t	rx_transfer_in_progress Set to 1 if a receive transfer is in progress, 0 otherwise.
uint8_t	data_bytes:2 1 byte for 7 or 8 bit data, 2 bytes for 9 bit data
uint8_t	bitrate_modulation:1 1 if bit rate modulation is enabled, 0 otherwise
uint32_t	open Used to determine if the channel is configured.
transfer_instance_t const *	p_transfer_rx
transfer_instance_t const *	p_transfer_tx
uint8_t const *	p_tx_src
uint32_t	tx_src_bytes
IRQn_Type	rxi_irq Receive IRQ number.
IRQn_Type	txi_irq Transmit IRQ number.
IRQn_Type	tei_irq Transmit end IRQ number.
IRQn_Type	eri_irq

	Error IRQ number.
void(*	p_callback)(uart_callback_args_t *p_args) Pointer to callback function.
void const *	p_context Pointer to user interrupt context data.
void *	p_reg Base register for this channel.
void(*	p_extpin_ctrl)(uint32_t channel, uint32_t level) External pin control.
uint32_t	baud_rate_error_x_1000 Baud rate <Maximum percent error> x 1000. baud_rate_error must be greater than 0 and less than 15000.
uint8_t *	p_rx_dst Destination buffer initialized by read() API.
uint32_t	rx_dst_bytes Number of bytes expected by the read() API.
volatile uint32_t	rx_bytes_count Number of bytes received since the last read()
uart_mode_t	uart_comm_mode UART communication mode selection.
uart_rs485_type_t	uart_rs485_mode UART RS485 communication channel type selection.
ioport_port_pin_t	rs485_de_pin

	UART Driver Enable pin.

Detailed Description

UART instance control block.

Field Documentation

◆ p_transfer_rx

transfer_instance_t const* sci_uart_instance_ctrl_t::p_transfer_rx
Optional transfer instance used to send or receive multiple bytes without interrupts.

◆ p_transfer_tx

transfer_instance_t const* sci_uart_instance_ctrl_t::p_transfer_tx
Optional transfer instance used to send or receive multiple bytes without interrupts.

◆ p_tx_src

uint8_t const* sci_uart_instance_ctrl_t::p_tx_src
Source buffer pointer used to fill hardware FIFO from transmit ISR.

◆ tx_src_bytes

uint32_t sci_uart_instance_ctrl_t::tx_src_bytes
Size of source buffer pointer used to fill hardware FIFO from transmit ISR.

The documentation for this struct was generated from the following file:

- r_sci_uart.h

uart_on_sci_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » UART on SCI

#include <r_sci_uart.h>

Data Fields

sci_clk_src_t	clk_src Use SCI_CLK_SRC_INT/EXT8X/EXT16X.
bool	baudclk_out Baud rate clock output enable.
bool	rx_edge_start Start reception on falling edge.
bool	noisecancel_en Noise cancel enable.
sci_uart_rx_fifo_trigger_t	rx_fifo_trigger
void(*	p_extpin_ctrl)(uint32_t channel, uint32_t level)
bool	bitrate_modulation Bitrate Modulation Function enable or disable.
uint32_t	baud_rate_error_x_1000 Baud rate <Maximum percent error> x 1000. baud_rate_error must be greater than 0 and less than 15000.
uart_mode_t	uart_comm_mode UART communication mode selection.
uart_rs485_type_t	uart_rs485_mode UART RS485 communication channel type selection.
ioport_port_pin_t	rs485_de_pin UART Driver Enable pin.

Detailed Description

UART on SCI device Configuration

Field Documentation

◆ p_extpin_ctrl

void(* uart_on_sci_cfg_t::p_extpin_ctrl) (uint32_t channel, uint32_t level)
Pointer to the user callback function to control external GPIO pin control used as RTS signal. **Parameters** [in] channel — The UART channel used. [in] level — When level is 0, assert RTS. When level is 1, deassert RTS.

◆ rx_fifo_trigger

sci_uart_rx_fifo_trigger_t uart_on_sci_cfg_t::rx_fifo_trigger
Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.

The documentation for this struct was generated from the following file:

- r_sci_uart.h

5.1.5.54 Sigma Delta ADC (SDADC)

Renesas Synergy Software Package Reference » HAL Layer

Driver for the 24-bit Sigma Delta A/D Converter (SDADC). More...

Data Structures

struct	sdadc_calibrate_args_t
struct	sdadc_channel_cfg_t
struct	adc_on_sdadc_cfg_t
struct	sdadc_instance_ctrl_t

Macros

#define	SDADC_CODE_VERSION_MAJOR (2U)

Enumerations

enum	sdadc_vref_src_t { SDADC_VREF_SRC_INTERNAL = 0, SDADC_VREF_SRC_EXTERNAL = 1 }
enum	sdadc_vref_voltage_t { SDADC_VREF_VOLTAGE_800_MV = 0, SDADC_VREF_VOLTAGE_1000_MV = 1, SDADC_VREF_VOLTAGE_1200_MV = 2, SDADC_VREF_VOLTAGE_1400_MV = 3, SDADC_VREF_VOLTAGE_1600_MV = 4, SDADC_VREF_VOLTAGE_1800_MV = 5, SDADC_VREF_VOLTAGE_2000_MV = 6, SDADC_VREF_VOLTAGE_2200_MV = 7, SDADC_VREF_VOLTAGE_2400_MV = 15 }
enum	sdadc_channel_input_t { SDADC_CHANNEL_INPUT_DIFFERENTIAL = 0, SDADC_CHANNEL_INPUT_SINGLE_ENDED = 1 }
enum	sdadc_channel_stage_1_gain_t { SDADC_CHANNEL_STAGE_1_GAIN_1 = 0, SDADC_CHANNEL_STAGE_1_GAIN_2 = 1, SDADC_CHANNEL_STAGE_1_GAIN_3 = 2, SDADC_CHANNEL_STAGE_1_GAIN_4 = 3, SDADC_CHANNEL_STAGE_1_GAIN_8 = 4 }
enum	sdadc_channel_stage_2_gain_t { SDADC_CHANNEL_STAGE_2_GAIN_1 = 0, SDADC_CHANNEL_STAGE_2_GAIN_2 = 1, SDADC_CHANNEL_STAGE_2_GAIN_4 = 2, SDADC_CHANNEL_STAGE_2_GAIN_8 = 3 }
enum	sdadc_channel_oversampling_t { SDADC_CHANNEL_OVERSAMPLING_64 = 0, SDADC_CHANNEL_OVERSAMPLING_128 = 1, SDADC_CHANNEL_OVERSAMPLING_256 = 2, SDADC_CHANNEL_OVERSAMPLING_512 = 3, SDADC_CHANNEL_OVERSAMPLING_1024 = 4, SDADC_CHANNEL_OVERSAMPLING_2048 = 5 }
enum	sdadc_channel_polarity_t { SDADC_CHANNEL_POLARITY_POSITIVE = 0, SDADC_CHANNEL_POLARITY_NEGATIVE = 1 }
enum	sdadc_channel_average_t { SDADC_CHANNEL_AVERAGE_NONE = 0, SDADC_CHANNEL_AVERAGE_8 = 12, SDADC_CHANNEL_AVERAGE_16 = 13, SDADC_CHANNEL_AVERAGE_32 = 14, SDADC_CHANNEL_AVERAGE_64 = 15 }

enum	sdadc_channel_inversion_t { SDADC_CHANNEL_INVERSION_OFF = 0, SDADC_CHANNEL_INVERSION_ON = 1 }
enum	sdadc_channel_count_formula_t { SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL = 0, SDADC_CHANNEL_COUNT_FORMULA_LINEAR = 1 }
enum	sdadc_calibration_t { SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET = 0, SDADC_CALIBRATION_EXTERNAL_OFFSET = 1, SDADC_CALIBRATION_EXTERNAL_GAIN = 2 }

Functions

ssp_err_t	R_SDADC_Open (adc_ctrl_t *p_api_ctrl, adc_cfg_t const *const p_cfg)
ssp_err_t	R_SDADC_ScanConfigure (adc_ctrl_t *p_api_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
ssp_err_t	R_SDADC_SampleStateCountSet (adc_ctrl_t *p_api_ctrl, adc_sample_state_t *p_sample)
ssp_err_t	R_SDADC_InfoGet (adc_ctrl_t *p_api_ctrl, adc_info_t *p_adc_info)
ssp_err_t	R_SDADC_ScanStart (adc_ctrl_t *p_api_ctrl)
ssp_err_t	R_SDADC_ScanStop (adc_ctrl_t *p_api_ctrl)
ssp_err_t	R_SDADC_CheckScanDone (adc_ctrl_t *p_api_ctrl)
ssp_err_t	R_SDADC_Read (adc_ctrl_t *p_api_ctrl, adc_register_t const reg_id, uint16_t *const p_data)
ssp_err_t	R_SDADC_Read32 (adc_ctrl_t *p_api_ctrl, adc_register_t const reg_id, uint32_t *const p_data)
ssp_err_t	R_SDADC_OffsetSet (adc_ctrl_t *const p_api_ctrl, adc_register_t const reg_id, int32_t const offset)
ssp_err_t	R_SDADC_Calibrate (adc_ctrl_t *const p_api_ctrl, void *const p_extend)
ssp_err_t	R_SDADC_Close (adc_ctrl_t *p_api_ctrl)
ssp_err_t	R_SDADC_VersionGet (ssp_version_t *const p_version)

Detailed Description

Driver for the 24-bit Sigma Delta A/D Converter (SDADC).

This module supports the SDADC peripheral. It implements the following interfaces:

- ADC Interface

Macro Definition Documentation

◆ SDADC_CODE_VERSION_MAJOR

#define SDADC_CODE_VERSION_MAJOR (2U)
Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ sdadc_calibration_t

enum sdadc_calibration_t	
Calibration mode.	
Enumerator	
SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET	Use internal reference to calibrate offset and gain.
SDADC_CALIBRATION_EXTERNAL_OFFSET	Use external reference to calibrate offset.
SDADC_CALIBRATION_EXTERNAL_GAIN	Use external reference to calibrate gain.

◆ sdadc_channel_average_t

enum sdadc_channel_average_t	
Per channel number of conversions to average before conversion end callback.	
Enumerator	
SDADC_CHANNEL_AVERAGE_NONE	Do not average (callback for each conversion)
SDADC_CHANNEL_AVERAGE_8	Average 8 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_16	Average 16 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_32	Average 32 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_64	Average 64 samples for each conversion end callback.

◆ sdadc_channel_count_formula_t

enum sdadc_channel_count_formula_t	
Select a formula to specify the number of conversions. The following symbols are used in the formulas: • N: Number of conversions • n: sdadc_channel_cfg_t::coefficient_n, do not set to 0 if m is 0 • m: sdadc_channel_cfg_t::coefficient_m, do not set to 0 if n is 0 Either m or n must be non-zero.	
Enumerator	
SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL	N = 32 * (2 ^ n - 1) + m * 2 ^ n.
SDADC_CHANNEL_COUNT_FORMULA_LINEAR	N = (32 * n) + m.

◆ sdadc_channel_input_t

enum sdadc_channel_input_t	
Per channel input mode.	
Enumerator	
SDADC_CHANNEL_INPUT_DIFFERENTIAL	Differential input.
SDADC_CHANNEL_INPUT_SINGLE_ENDED	Single-ended input.

◆ sdadc_channel_inversion_t

enum sdadc_channel_inversion_t	
Per channel polarity, valid for negative-side single-ended input only.	
Enumerator	
SDADC_CHANNEL_INVERSION_OFF	Do not invert conversion result.
SDADC_CHANNEL_INVERSION_ON	Invert conversion result.

◆ sdadc_channel_oversampling_t

enum sdadc_channel_oversampling_t	
Per channel oversampling ratio.	
Enumerator	
SDADC_CHANNEL_OVERSAMPLING_64	Oversampling ratio of 64.
SDADC_CHANNEL_OVERSAMPLING_128	Oversampling ratio of 128.
SDADC_CHANNEL_OVERSAMPLING_256	Oversampling ratio of 256.
SDADC_CHANNEL_OVERSAMPLING_512	Oversampling ratio of 512.
SDADC_CHANNEL_OVERSAMPLING_1024	Oversampling ratio of 1024.
SDADC_CHANNEL_OVERSAMPLING_2048	Oversampling ratio of 2048.

◆ sdadc_channel_polarity_t

enum sdadc_channel_polarity_t	
Per channel polarity, valid for single-ended input only.	
Enumerator	
SDADC_CHANNEL_POLARITY_POSITIVE	Positive-side single-ended input.
SDADC_CHANNEL_POLARITY_NEGATIVE	Negative-side single-ended input.

◆ sdadc_channel_stage_1_gain_t

enum sdadc_channel_stage_1_gain_t	
Per channel stage 1 gain options.	
Enumerator	
SDADC_CHANNEL_STAGE_1_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_1_GAIN_2	Gain of 2.
SDADC_CHANNEL_STAGE_1_GAIN_3	Gain of 3 (only valid for stage 1)
SDADC_CHANNEL_STAGE_1_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_1_GAIN_8	Gain of 8.

◆ sdadc_channel_stage_2_gain_t

enum sdadc_channel_stage_2_gain_t	
Per channel stage 2 gain options.	
Enumerator	
SDADC_CHANNEL_STAGE_2_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_2_GAIN_2	Gain of 2.
SDADC_CHANNEL_STAGE_2_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_2_GAIN_8	Gain of 8.

◆ sdadc_vref_src_t

enum sdadc_vref_src_t	
Source of Vref.	
Enumerator	
SDADC_VREF_SRC_INTERNAL	Vref is internally sourced, can be output as SBIAS.
SDADC_VREF_SRC_EXTERNAL	Vref is externally sourced from the VREFI pin.

◆ sdadc_vref_voltage_t

enum sdadc_vref_voltage_t	
Voltage of Vref.	
Enumerator	
SDADC_VREF_VOLTAGE_800_MV	Vref is 0.8 V.
SDADC_VREF_VOLTAGE_1000_MV	Vref is 1.0 V.
SDADC_VREF_VOLTAGE_1200_MV	Vref is 1.2 V.
SDADC_VREF_VOLTAGE_1400_MV	Vref is 1.4 V.
SDADC_VREF_VOLTAGE_1600_MV	Vref is 1.6 V.
SDADC_VREF_VOLTAGE_1800_MV	Vref is 1.8 V.
SDADC_VREF_VOLTAGE_2000_MV	Vref is 2.0 V.
SDADC_VREF_VOLTAGE_2200_MV	Vref is 2.2 V.
SDADC_VREF_VOLTAGE_2400_MV	Vref is 2.4 V (only valid for external Vref)

Function Documentation

◆ R_SDADC_Calibrate()

ssp_err_t R_SDADC_Calibrate (adc_ctrl_t *const *p_api_ctrl*, void *const *p_extend*)

Requires sdadc_calibrate_args_t passed to p_extend. Calibrates the specified channel. Calibration is not required or supported for single-ended mode. Internal calibration is automatically done during open() unless it is disabled in the user configuration. This API is provided primarily for external calibration. A callback with the event ADC_EVENT_CALIBRATION_COMPLETE is called when calibration completes. The calibration status can also be monitored using adc_api_t::statusGet(). Implements adc_api_t::calibrate().

During external offset calibration, apply a differential voltage of 0 to ANSDnP - ANSDnN, where n is the input channel and ANSDnP is OPAMP0 for channel 4 and ANSDnN is OPAMP1 for channel 4. Complete external offset calibration before external gain calibration.

During external gain calibration apply a voltage between 0.4 V / total_gain and 0.8 V / total_gain. The differential voltage applied during calibration is corrected to a conversion result of 0x7FFFFF, which is the maximum possible positive differential measurement.

This function clears the offset value. If offset is required after calibration, it must be reapplied after calibration is complete using adc_api_t::offsetSet.

Return values

SSP_SUCCESS	Calibration began successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Calibration cannot be performed if conversion or calibration is already in progress.

Select software trigger.

Select single scan mode.

Enable calibration.

Set the offset voltage to 0 mV.

Set the calibration mode.

Start calibration.

Confirm that calibration started.

Return the error code

◆ R_SDADC_CheckScanDone()

ssp_err_t R_SDADC_CheckScanDone (adc_ctrl_t * *p_api_ctrl*)

Returns the status of a scan started by software, including calibration scans. It is not possible to determine the status of a scan started by a hardware trigger. Implements adc_api_t::scanStatusGet().

Return values

SSP_SUCCESS	No software scan or calibration is in progress.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

If calibration is in progress, return an error.

Return the scan status of a scan triggered by software.

◆ R_SDADC_Close()

ssp_err_t R_SDADC_Close (adc_ctrl_t * *p_api_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the SDADC peripheral.
Implements adc_api_t::close().

Return values

SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Perform parameter checking

Mark driver as closed

Disable hardware triggers.

Stop A/D conversion.

Disable interrupts.

Wait 3 us in normal mode as required by the hardware manual.

Turn off the power of VBIAS, channel, and sigma-delta A/D converter.

Turn off the power of ADBGR, SBIAS, and ADREG.

Stop the input clock for the 24-bit sigma-delta A/D converter (SDADCCLK).

Enter the module-stop state.

Release the hardware lock

Return the error code

◆ R_SDADC_InfoGet()

ssp_err_t R_SDADC_InfoGet (adc_ctrl_t * *p_api_ctrl*, adc_info_t * *p_adc_info*)

Returns the address of the lowest number configured channel, the total number of results to be read in order to read the results of all configured channels, the size of each result, and the ELC event enumerations. Implements adc_api_t::infoGet().

Return values

SSP_SUCCESS	Information stored in p_adc_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Retrieve the scan mask of active channels from the control block

Determine the lowest channel that is configured.

Determine the highest channel that is configured.

Determine the size of data that must be read to read all the channels between and including the highest and lowest channels.

Determine the base address and transfer size.

Specify the peripheral name in the ELC list

Set sensor information to invalid value

◆ R_SDADC_OffsetSet()

ssp_err_t R_SDADC_OffsetSet (adc_ctrl_t *const *p_api_ctrl*, adc_register_t const *reg_id*, int32_t const *offset*)

Sets the offset. Offset is applied after stage 1 of the input channel. Offset can only be applied when the channel is configured for differential input. Implements adc_api_t::offsetSet().

Note: The offset is cleared if adc_api_t::calibrate() is called. The offset can be re-applied if necessary after the the callback with event ADC_EVENT_CALIBRATION_COMPLETE is called.

Parameters

[in]	p_api_ctrl	See p_ctrl in adc_api_t::offsetSet().
[in]	reg_id	See reg_id in adc_api_t::offsetSet().
[in]	offset	Must be between -15 and 15, offset (mV) = 10.9376 mV * offset_steps / stage 1 gain.

Return values

SSP_SUCCESS	Offset updated successfully.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Offset cannot be updated if conversion or calibration is in progress.

Set the offset.

◆ R_SDADC_Open()

ssp_err_t R_SDADC_Open (adc_ctrl_t * *p_api_ctrl*, adc_cfg_t const *const *p_cfg*)

Applies power to the SDADC and initializes the hardware based on the user configuration. As part of this initialization, the SDADC clock is configured and enabled. If an interrupt priority is non-zero, enables an interrupt which will call a callback to notify the user when a conversion, scan, or calibration is complete. For all channels that are configured in differential mode, calibration is performed unless it is disabled in the user configuration. Implements adc_api_t::open().

Return values

SSP_SUCCESS	Configuration and calibration successful.
SSP_ERR_CALIBRATE_FAILED	Calibration failed.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.
SSP_ERR_IN_USE	Control block is already open.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt is disabled

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Ensure the SDADC exists on the hardware, enable interrupts, and reserve the SDADC hardware lock.

Initialize the hardware based on the configuration.

Start the SDADC.

Configure the SDADC clock source and divisor to the clock source selected in the clock configuration.

Set the reference voltage for sensors (internal or external VREF mode).

Set the A/D conversion operation mode (normal or low power mode).

Supply the 24-bit sigma-delta A/D converter clock (SDADCCLK).

Turn on the power of ADBGR, SBIAS, and ADREG.

Stabilization wait time of 2 ms is required between power on of ADBGR/SBIAS/ADREG and VBIAS/channel/SDADC.

Turn on the power of VBIAS, channel, and sigma-delta A/D converter.

For each channel:

1. Set the oversampling ratio.
2. Set the gain.
3. Select single-end input/differential input.
4. If differential mode is used and calibration during open is not skipped, enable calibration on this channel.
5. Select the polarity of single-end input (only for single-end mode).

Set the A/D conversion count.

1. Select the averaging operation. Select the average data count.
    a. Select whether to reverse the A/D conversion results of single-end input (only for negative-side single end mode).
2. Enable conversion for the channel.

Configure enabled channels and autoscan mode.

If the A/D conversion trigger is ELC hardware events, the hardware events are enabled in adc_api_t::scanStart().

Mark driver as open by initializing it to "SDAD" - its ASCII equivalent.

If at least one channel is configured for differential mode and calibration is not disabled in the user configuration, calibrate all PGAs configured for differential mode.

The calibration takes approximately 3.4 ms per channel in normal mode or 27 ms in low power mode. The open() API blocks waiting for calibration to complete unless calibration is skipped. Calibration during open can be skipped and handled in at the application level by setting adc_on_sdadc_cfg_t::skip_internal_calibration to true, then calling adc_api_t::calibrate().

◆ R_SDADC_Read()

ssp_err_t R_SDADC_Read (adc_ctrl_t * *p_api_ctrl*, adc_register_t const *reg_id*, uint16_t *const *p_data*)

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. Implements adc_api_t::read().

Return values

SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Read the most recent conversion value into a 16-bit result.

◆ R_SDADC_Read32()

ssp_err_t R_SDADC_Read32 (adc_ctrl_t * *p_api_ctrl*, adc_register_t const *reg_id*, uint32_t *const *p_data*)

Reads the most recent conversion result from a channel. Implements adc_api_t::read32().

Return values

SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Read the most recent conversion value into a 32-bit result.

◆ R_SDADC_SampleStateCountSet()

ssp_err_t R_SDADC_SampleStateCountSet (adc_ctrl_t * *p_api_ctrl*, adc_sample_state_t * *p_sample*)

adc_api_t::sampleStateCountSet is not supported on the SDADC.

Return values

SSP_ERR_UNSUPPORTED	This API is not supported.

Return the unsupported error.

◆ R_SDADC_ScanConfigure()

ssp_err_t R_SDADC_ScanConfigure (adc_ctrl_t * *p_api_ctrl*, adc_channel_cfg_t const *const *p_channel_cfg*)

Configures the enabled channels of the ADC. Only adc_channel_cfg_t::scan_mask is used. Implements adc_api_t::scanCfg().

Return values

SSP_SUCCESS	Information stored in p_adc_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.

Update the enabled channels.

◆ R_SDADC_ScanStart()

ssp_err_t R_SDADC_ScanStart (adc_ctrl_t * *p_api_ctrl*)

If the SDADC is configured for hardware triggers, enables hardware triggers. Otherwise, starts a scan. Implements adc_api_t::scanStart().

Return values

SSP_SUCCESS	Scan started or hardware triggers enabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_IN_USE	A conversion or calibration is in progress.

Conversion cannot be performed if conversion or calibration is already in progress.

If the unit is configured for software triggering, trigger a scan.

Otherwise, enable hardware triggers.

Return the error code

◆ R_SDADC_ScanStop()

ssp_err_t R_SDADC_ScanStop (adc_ctrl_t * *p_api_ctrl*)

If the SDADC is configured for hardware triggers, disables hardware triggers. Otherwise, stops any in-progress scan started by software. Implements adc_api_t::scanStop().

Return values

SSP_SUCCESS	Scan stopped or hardware triggers disabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

If the unit is configured for software triggering, stop the scan.

Otherwise, disable hardware triggers.

Return the error code

◆ R_SDADC_VersionGet()

ssp_err_t R_SDADC_VersionGet (ssp_version_t *const *p_version*)
Gets the API and code version. Implements adc_api_t::versionGet().

Return values

SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Return the version number

sdadc_calibrate_args_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Sigma Delta ADC (SDADC)

#include <r_sdadc.h>

Data Fields

adc_register_t	channel Which channel to calibrate.
sdadc_calibration_t	mode Calibration mode.

Detailed Description

Structure to pass to the adc_api_t::calibrate p_extend argument.

The documentation for this struct was generated from the following file:

- r_sdadc.h

sdadc_channel_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Sigma Delta ADC (SDADC)

#include <r_sdadc.h>

Data Fields

sdadc_channel_stage_2_gain_t	stage_2_gain:2 Gain of PGA stage 2, must be 1 for single-ended input.
sdadc_channel_stage_1_gain_t	stage_1_gain:3 Gain of PGA stage 1, must be 1 for single-ended input.
sdadc_channel_oversampling_t	oversampling:3 Oversampling ratio, must be 256 in single-ended input.
sdadc_channel_polarity_t	polarity:1 Polarity, valid for single-ended mode only.
sdadc_channel_input_t	input:1 Single-ended or differential input.
uint32_t	coefficient_m:5 See sdadc_channel_count_formula_t.
uint32_t	coefficient_n:3 See sdadc_channel_count_formula_t.
sdadc_channel_average_t	average:4 Number of samples to average for each conversion result.
sdadc_channel_inversion_t	invert:1 Whether to invert negative single-ended input.
sdadc_channel_count_formula_t	count_formula:1 Linear or exponential formula used for number of conversions.

Detailed Description

SDADC per channel configuration.

The documentation for this struct was generated from the following file:

- r_sdadc.h

adc_on_sdadc_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Sigma Delta ADC (SDADC)

#include <r_sdadc.h>

Data Fields

uint8_t	calibration_end_ipl Calibration end interrupt priority.
uint8_t	conversion_end_ipl Conversion end interrupt priority.
bool	skip_internal_calibration Whether to skip internal calibration of of the PGA during open.
sdadc_vref_src_t	vref_src Source of Vref (internal or external)
sdadc_vref_voltage_t	vref_voltage
sdadc_channel_cfg_t const *	p_channel_cfgs [SDADC_MAX_NUM_CHANNELS] Configuration for each channel, set to NULL if unused.

Detailed Description

SDADC configuration extension. This extension is required and must be provided in adc_cfg_t::p_extend.

Field Documentation

◆ vref_voltage

sdadc_vref_voltage_t adc_on_sdadc_cfg_t::vref_voltage
Voltage of Vref, required for both internal and external Vref. If Vref is from an external source, the voltage must match the specified voltage within 3%.

The documentation for this struct was generated from the following file:

- r_sdadc.h

sdadc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » Sigma Delta ADC (SDADC)

#include <r_sdadc.h>

Data Fields

adc_mode_t	mode Single scan or continuous mode.
adc_resolution_t	resolution 16 or 24 bit resolution
adc_alignment_t	alignment Left or right alignment.
void const *	p_context Placeholder for user data.
R_SDADC0_Type *	p_reg Base register for this unit.
void(*	p_callback)(adc_callback_args_t *p_args) User callback pointer.

adc_trigger_t	trigger Software or hardware trigger.
uint32_t	trigger_enabled If set, hardware trigger was enabled before calibration.
uint32_t	opened Boolean to verify that the Unit has been initialized.
uint32_t	scan_mask Scan mask of enabled channels.
uint32_t	scan_cfg_mask Scan mask of configured channels.
uint16_t	unit SDADC Unit in use.
volatile uint8_t	calib_status Calibration in progress if set.
IRQn_Type	scan_end_irq Scan end IRQ number.
IRQn_Type	calib_end_irq Calibration end IRQ number.
IRQn_Type	conv_end_irq Conversion end IRQ number.

Detailed Description

ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().

The documentation for this struct was generated from the following file:

- r_sdadc.h

5.1.5.55 SDMMC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the SD/MMC Host Interface (SDHI). More...

Data Structures

struct	sdmmc_instance_ctrl_t
struct	sdmmc_extended_cfg_t

Enumerations

enum	sdmmc_card_detect_t { SDMMC_CARD_DETECT_NONE, SDMMC_CARD_DETECT_CD }
enum	sdmmc_write_protect_t { SDMMC_WRITE_PROTECT_NONE, SDMMC_WRITE_PROTECT_WP }

Functions

ssp_err_t	R_SDMMC_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg)
ssp_err_t	R_SDMMC_Close (sdmmc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SDMMC_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t	R_SDMMC_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
ssp_err_t	R_SDMMC_Control (sdmmc_ctrl_t *const p_api_ctrl, ssp_command_t const command, void *p_data)
ssp_err_t	R_SDMMC_ReadIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
ssp_err_t	R_SDMMC_WriteIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
ssp_err_t	R_SDMMC_ReadIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const

	p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
ssp_err_t	R_SDMMC_WriteIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
ssp_err_t	R_SDMMC_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable)
ssp_err_t	R_SDMMC_VersionGet (ssp_version_t *const p_version)
ssp_err_t	R_SDMMC_InfoGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_info_t *const p_info)
ssp_err_t	R_SDMMC_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Detailed Description

Driver for the SD/MMC Host Interface (SDHI).

SD/MMC driver to access SD, eMMC, and SDIO devices.

Enumeration Type Documentation

◆ sdmmc_card_detect_t

enum sdmmc_card_detect_t	
Card detection configuration options.	
Enumerator	
SDMMC_CARD_DETECT_NONE	Card detection unused.
SDMMC_CARD_DETECT_CD	Card detection using the SDnCD pin.

◆ sdmmc_write_protect_t

enum sdmmc_write_protect_t	
Write protection configuration option	
Enumerator	
SDMMC_WRITE_PROTECT_NONE	Write protection unused.
SDMMC_WRITE_PROTECT_WP	write protection using SDnWP pin

Function Documentation

◆ R_SDMMC_Close()

ssp_err_t R_SDMMC_Close (sdmmc_ctrl_t *const *p_api_ctrl*)

Closes an open SD/MMC device. Implements sdmmc_api_t::close().

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Disable SDHI interrupts.

Close the transfer driver.

Turn on module stop bit (turn module off).

Release hardware lock.

◆ R_SDMMC_Control()

ssp_err_t R_SDMMC_Control (sdmmc_ctrl_t *const *p_api_ctrl*, ssp_command_t const *command*, void * *p_data*)

Sends control commands to and receives the status of the SD/MMC device. Implements sdmmc_api_t::control().

Return values

SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	Command is invalid.
SSP_ERR_INVALID_SIZE	Block size not in valid range of 1-512 for SDIO or 512 only for SD cards and eMMC.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Get the command status and return to called function.

◆ R_SDMMC_Erase()

ssp_err_t R_SDMMC_Erase (sdmmc_ctrl_t *const *p_api_ctrl*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Erases sectors of an SD card or eMMC device. Implements sdmmc_api_t::erase().

This function blocks until erase is complete.

Return values

SSP_SUCCESS	Erase operation requested.
SSP_ERR_ASSERTION	A required pointer is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_PROTECTED	SD card is Write Protected.
SSP_ERR_ERASE_FAILED	Erase operation unsuccessful.

Note

This function is reentrant for different channels.

Send command to set start erase address (CMD35 for eMMC, CMD32 for SD).

Send command to set end erase address (CMD36 for eMMC, CMD33 for SD).

Send erase command (CMD38).

◆ R_SDMMC_InfoGet()

ssp_err_t R_SDMMC_InfoGet (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_info_t *const *p_info*)

Provides information about the connected device and driver status. Implements sdmmc_api_t::infoGet().

Return values

SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_NOT_OPEN	Driver has not been initialized.

Note

This function is reentrant.

Copy information stored during open.

Check if the card is busy.

◆ R_SDMMC_IoIntEnable()

ssp_err_t R_SDMMC_IoIntEnable (sdmmc_ctrl_t *const *p_api_ctrl*, bool *enable*)

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

Return values

SSP_SUCCESS	Card enabled or disabled SDIO interrupts successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.

Note

This function is reentrant for different channels.

Make sure the card is not busy.

Enable or disable interrupt.

◆ R_SDMMC_Open()

ssp_err_t R_SDMMC_Open (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_cfg_t const *const *p_cfg*)

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This API blocks until all identification and configuration commands are complete.

For SDIO, SDIO interrupts are enabled after card identification is complete. SDIO interrupts can be disabled using sdmmc_api_t::ioIntEnable().

Implements sdmmc_api_t::open().

Return values

SSP_SUCCESS	Port is available and is now open for read/write/control access.
SSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512.
SSP_ERR_INVALID_ARGUMENT	Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only.
SSP_ERR_ALREADY_OPEN	Driver has already been opened with this instance of the control structure.
SSP_ERR_HW_LOCKED	The channel specified has already been opened.
SSP_ERR_CARD_INIT_FAILED	Hardware related failure occurred, with the MCU or with the card itself.
SSP_ERR_IRQ_BSP_DISABLED	Access interrupt is not enabled.

SSP_ERR_CARD_NOT_INSERTED	Card detection is enabled and no card is plugged in.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- cgc_api_t::systemClockFreqGet

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Verify the requested hardware channel exists on the MCU.

Configure interrupts.

Acquire lock before changing vector table or p_ctrl.

Turn off module stop bit (turn module on).

Perform the identification procedure for SD card or eMMC device.

Configure bus clock, block size, and bus width.

Check to see if the card is write protected (SD cards only).

◆ R_SDMMC_Read()

ssp_err_t R_SDMMC_Read (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements sdmmc_api_t::read().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Configure the transfer interface for reading.

Read data from SD or eMMC device.

◆ R_SDMMC_ReadIo()

ssp_err_t R_SDMMC_ReadIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*)

The Read function reads a one byte register from an SDIO card. Implements sdmmc_api_t::readIo().

This function blocks until the command is sent and the response is received. p_data contains the register value read when this function returns.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Call SDMMC protocol read function

◆ R_SDMMC_ReadIoExt()

ssp_err_t R_SDMMC_ReadIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *function*, uint32_t const *address*, uint32_t *const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Reads data from an SDIO card function. Implements sdmmc_api_t::readIoExt().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Return values

SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

Configure the transfer interface for reading.

Read data from SDIO device.

◆ R_SDMMC_VersionGet()

ssp_err_t R_SDMMC_VersionGet (ssp_version_t *const *p_version*)

Returns the version of the firmware and API. Implements sdmmc_api_t::versionGet().

Return values

SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	Null Pointer.

Note

This function is reentrant.

◆ R_SDMMC_Write()

ssp_err_t R_SDMMC_Write (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements sdmmc_api_t::write().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written.

Return values

SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	Handle or Source address is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_PROTECTED	SD card is Write Protected.
SSP_ERR_WRITE_FAILED	Write operation failed.

Note

This function is reentrant for different channels.

Configure the transfer interface for writing.

Call SDMMC protocol write function

Write data to SD or eMMC device.

◆ R_SDMMC_WriteIo()

ssp_err_t R_SDMMC_WriteIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*, sdmmc_io_write_mode_t const *read_after_write*)

Writes a one byte register to an SDIO card. Implements sdmmc_api_t::writeIo().

This function blocks until the command is sent and the response is received. The register has been written when this function returns. If read_after_write is true, p_data contains the register value read when this function returns.

Return values

SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	Handle or Source address is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_FAILED	Write operation failed.

Note

This function is reentrant for different channels.

Call SDMMC protocol write function with valid parameters

◆ R_SDMMC_WriteIoExt()

ssp_err_t R_SDMMC_WriteIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *function*, uint32_t const *address*, uint32_t const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Writes data to an SDIO card function. Implements sdmmc_api_t::writeIoExt().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written.

Return values

SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_FAILED	Write operation failed.

Note

This function is reentrant for different channels.

Configure the transfer interface for writing.

Write data to SDIO device.

sdmmc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SDMMC

#include <r_sdmmc.h>

Data Fields

uint32_t	open used to determine if channel is open
sdmmc_hw_t	hw Temporary storage for channel, media type and bus width.
transfer_instance_t const *	p_lower_lvl_transfer

	Transfer instance used to transfer data with DMA or DTC.
sdmmc_info_t	status To load various status information.
bool	transfer_in_progress Flag to detect transfer status.
bool	write_protect write protect status
void(*	p_callback)(sdmmc_callback_args_t *p_args) User callback pointer.
void const *	p_context Placeholder for user data.
R_SDHI0_Type *	p_reg Base register information.
volatile sdhi_event_t	sdhi_event Update event status.
IRQn_Type	transfer_irq Scan end IRQ number.
sdmmc_transfer_dir_t	transfer_dir Info on read or write operation in progress.
uint8_t *	p_transfer_data Temporary storage for transfer data.
uint32_t	transfer_blocks_total Total transfer block count.

uint32_t	transfer_block_current Transfer current block.
uint32_t	transfer_block_size Transfer block size.
uint32_t	aligned_buff [SDMMC_MAX_BLOCK_SIZE/sizeof(uint32_t)] Aligned buffer.

Detailed Description

SDMMC instance control block. This is private to the SSP and should not be used or modified by the application.

The documentation for this struct was generated from the following file:

- r_sdmmc.h

sdmmc_extended_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SDMMC

#include <r_sdmmc.h>

Data Fields

uint32_t	block_size
sdmmc_card_detect_t	card_detect
sdmmc_write_protect_t	write_protect

Detailed Description

Extended SDMMC configuration, to be pointed to p_extend.

Field Documentation

◆ block_size

uint32_t sdmmc_extended_cfg_t::block_size
Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

◆ card_detect

sdmmc_card_detect_t sdmmc_extended_cfg_t::card_detect
Whether or not card detection is used.

◆ write_protect

sdmmc_write_protect_t sdmmc_extended_cfg_t::write_protect
Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

The documentation for this struct was generated from the following file:

- r_sdmmc.h

5.1.5.56 SLCDC

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Segment LCD Controller (SLCDC). More...

Data Structures

struct	slcdc_instance_ctrl_t

Functions

ssp_err_t	R_SLCDC_Open (slcdc_ctrl_t *const p_api_ctrl, slcdc_cfg_t const *const p_cfg)
ssp_err_t	R_SLCDC_Write (slcdc_ctrl_t *const p_api_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
ssp_err_t	R_SLCDC_Modify (slcdc_ctrl_t *const p_api_ctrl, slcdc_size_t const segment_number, slcdc_size_t const data, slcdc_size_t const

	data_mask)
ssp_err_t	R_SLCDC_Start (slcdc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SLCDC_Stop (slcdc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SLCDC_ContrastIncrease (slcdc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SLCDC_ContrastDecrease (slcdc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const p_api_ctrl, slcdc_display_area_t const display_area)
ssp_err_t	R_SLCDC_Close (slcdc_ctrl_t *const p_api_ctrl)
ssp_err_t	R_SLCDC_VersionGet (ssp_version_t *const p_version) Retrieve the API version number. More...

Detailed Description

Driver for the Segment LCD Controller (SLCDC).

Summary

Extends SLCDC Interface.

Function Documentation

◆ R_SLCDC_Close()

ssp_err_t R_SLCDC_Close (slcdc_ctrl_t *const *p_api_ctrl*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

Clear all the data segment registers.

Disable the LCD display area. Segment pin outputs de-select signal

Disable voltage circuit

Set voltage generator to External

Protect OFF of CGC.

Disable LCD clock

Protect ON of CGC.

Release hardware lock for SLCD

SLCDC Power off - enter module-stop state for the SLCDC

Mark control block close.

◆ R_SLCDC_ContrastDecrease()

ssp_err_t R_SLCDC_ContrastDecrease (slcdc_ctrl_t *const *p_api_ctrl*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_UNSUPPORTED	Unsupported operation

The VLCD setting is valid only when the voltage boost circuit is operating

Verify the new volt is within the range.

Stop the internal voltage boost/capacitor split circuit.

Set new voltage value.

Wait 5ms minimum as per HW manual

Enable the voltage boost circuit or capacitor split circuit.

◆ R_SLCDC_ContrastIncrease()

ssp_err_t R_SLCDC_ContrastIncrease (slcdc_ctrl_t *const *p_api_ctrl*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_UNSUPPORTED	Unsupported operation

The VLCD setting is valid only when the voltage boost circuit is operating

Verify the new volt is within the range.

Stop the internal voltage boost/capacitor split circuit.

Set new voltage value.

Wait 5ms minimum as per HW manual

Enable the voltage boost circuit or capacitor split circuit.

◆ R_SLCDC_Modify()

ssp_err_t R_SLCDC_Modify (slcdc_ctrl_t *const *p_api_ctrl*, slcdc_size_t const *segment_number*, slcdc_size_t const *data*, slcdc_size_t const *data_mask*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

Masks the data being displayed.

Specifies the data to rewrite.

◆ R_SLCDC_Open()

ssp_err_t R_SLCDC_Open (slcdc_ctrl_t *const *p_api_ctrl*, slcdc_cfg_t const *const *p_cfg*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_HW_LOCKED	SLCDC resource is locked.

Returns

See Common Error Codes for other possible return codes. This function calls
- fmi_api_t::productFeatureGet

Configure the SLCDC driver

Record the configuration on the device for later use

Mark control block state open so subsequent calls know the device is open.

◆ R_SLCDC_SetDisplayArea()

ssp_err_t R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const *p_api_ctrl*, slcdc_display_area_t const *display_area*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_UNSUPPORTED	Unsupported operation
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_NOT_ENABLED	RTC not enabled for blink operation

Returns

See Common Error Codes for other possible return codes. This function calls
- fmi_api_t::productFeatureGet

When the number of time slices is eight, LCD display data registers (A-pattern, B-pattern, or blinking display) cannot be selected.

Set the LCD display data area.

◆ R_SLCDC_Start()

ssp_err_t R_SLCDC_Start (slcdc_ctrl_t *const *p_api_ctrl*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

Enable the voltage boost circuit or capacitor split circuit.

Enable the LCD display.

◆ R_SLCDC_Stop()

ssp_err_t R_SLCDC_Stop (slcdc_ctrl_t *const *p_api_ctrl*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

Disable the LCD display

Disable the voltage boost circuit or capacitor split circuit.

◆ R_SLCDC_VersionGet()

ssp_err_t R_SLCDC_VersionGet (ssp_version_t *const *p_version*)

Retrieve the API version number.

Return values

SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ R_SLCDC_Write()

ssp_err_t R_SLCDC_Write (slcdc_ctrl_t *const *p_api_ctrl*, slcdc_size_t const *start_segment*, slcdc_size_t const *const *p_data*, slcdc_size_t const *segment_count*)

Return values

SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block structure and display data is NULL
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

Get the register address of the specified segment and write data to it.

slcdc_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SLCDC

#include <r_slcdc.h>

Data Fields

slcdc_display_state_t	state Status of SLCD module.
slcdc_cfg_t	info SLCDC config info.
void const *	p_context Pointer to the higher level device context.
R_LCD_Type *	p_reg Pointer to register base address.

Detailed Description

SLCDC control block. DO NOT INITIALIZE. Initialization occurs when slcdc_api_t::open is called

The documentation for this struct was generated from the following file:

- r_slcdc.h

5.1.5.57 SSI

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Serial Sound Interface (SSI). More...

Data Structures

struct	ssi_instance_ctrl_t
struct	i2s_on_ssi_cfg_t

Enumerations

enum	ssi_audio_clock_t { SSI_AUDIO_CLOCK_EXTERNAL = 0, SSI_AUDIO_CLOCK_GTIOC1A = 1 }

Functions

ssp_err_t	R_SSI_Open (i2s_ctrl_t *const p_api_ctrl, i2s_cfg_t const *const p_cfg) Opens the SSI. Implements i2s_api_t::open. More...
ssp_err_t	R_SSI_Stop (i2s_ctrl_t *const p_api_ctrl, i2s_dir_t const dir) Stops SSI. Implements i2s_api_t::stop. More...
ssp_err_t	R_SSI_Close (i2s_ctrl_t *const p_api_ctrl) Closes SSI. Implements i2s_api_t::close. More...
ssp_err_t	R_SSI_Write (i2s_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint16_t const bytes) Writes data buffer to SSI. Implements i2s_api_t::write. More...
ssp_err_t	R_SSI_Read (i2s_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint16_t const bytes) Reads data into provided buffer. Implements i2s_api_t::read. More...

ssp_err_t	R_SSI_WriteRead (i2s_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes) Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead. More...
ssp_err_t	R_SSI_Mute (i2s_ctrl_t *const p_api_ctrl, i2s_mute_t const mute_enable) Mutes SSI. Implements i2s_api_t::mute. More...
ssp_err_t	R_SSI_InfoGet (i2s_ctrl_t *const p_api_ctrl, i2s_info_t *const p_info) Get I2S information and store it in provided pointer p_info. Implements i2s_api_t::infoGet. More...
ssp_err_t	R_SSI_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...

Detailed Description

Driver for the Serial Sound Interface (SSI).

Summary

Extends I2S Interface.

Enumeration Type Documentation

◆ ssi_audio_clock_t

enum ssi_audio_clock_t	
Clock source. Selects GPT channel 1 or AUDIO_CLK input pin.	
Enumerator	
SSI_AUDIO_CLOCK_EXTERNAL	Audio clock source is the AUDIO_CLK input pin.
SSI_AUDIO_CLOCK_GTIOC1A	Audio clock source is internal connection to GPT channel 1 output.

Function Documentation

◆ R_SSI_Close()

ssp_err_t R_SSI_Close (i2s_ctrl_t *const *p_api_ctrl*)

Closes SSI. Implements i2s_api_t::close.

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

Return values

SSP_SUCCESS	Device closed successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Stop feeding clock to SSI peripheral to deactivate it.

If a timer instance is provided, close the timer instance.

If a transfer instance is provided for write, close the transfer instance.

If a transfer instance is provided for read, close the transfer instance.

Release HW lock.

◆ R_SSI_InfoGet()

ssp_err_t R_SSI_InfoGet (i2s_ctrl_t *const *p_api_ctrl*, i2s_info_t *const *p_info*)

Get I2S information and store it in provided pointer p_info. Implements i2s_api_t::infoGet.

Return values

SSP_SUCCESS	Information stored successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_info parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Get the SSI hardware status information.

If SSI hardware status is idle, set status to stopped. Otherwise, set status to in use.

Get the sampling frequency information.

◆ R_SSI_Mute()

ssp_err_t R_SSI_Mute (i2s_ctrl_t *const *p_api_ctrl*, i2s_mute_t const *mute_enable*)

Mutes SSI. Implements i2s_api_t::mute.

Data is still written while mute is enabled, but the transmit line outputs zeros.

Return values

SSP_SUCCESS	Transmission is muted.
SSP_ERR_ASSERTION	The pointer to p_ctrl was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Enables the mute if MUTE_ON is set. Otherwise, disables the mute.

◆ R_SSI_Open()

ssp_err_t R_SSI_Open (i2s_ctrl_t *const *p_api_ctrl*, i2s_cfg_t const *const *p_cfg*)

Opens the SSI. Implements i2s_api_t::open.

This function calculates the clock divisor based on the input audio clock frequency and the requested sampling frequency. It sets this clock divisor and the configurations specified in i2s_cfg_t . It also opens the timer and transfer interfaces if they are provided.

Return values

SSP_SUCCESS	Ready for I2S communication.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_cfg is null.
SSP_ERR_IN_USE	The requested channel has already been opened or hardware has been locked.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet
- fmi_api_t::eventInfoGet
- transfer_api_t::open
- timer_api_t::open

Configure dependent timer and transfer drivers.

Configure interrupts.

Mark driver as open by initializing it to "SSI" in its ASCII equivalent.

◆ R_SSI_Read()

ssp_err_t R_SSI_Read (i2s_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint16_t const *bytes*)

Reads data into provided buffer. Implements i2s_api_t::read.

This function resets the transfer if the transfer interface is used, or reads the length of data available in the FIFO then stores the remaining read buffer in the control block to be filled in the ISR.

Read() cannot be called if another write(), read() or writeRead() operation is in progress. Read can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

SSP_SUCCESS	Read initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_dest was null, or bytes requested was 0.
SSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset
- timer_api_t::start

If a transfer instance is provided for read, reset the transfer. Otherwise unload the receive FIFO.

Make sure reception is enabled.

◆ R_SSI_Stop()

ssp_err_t R_SSI_Stop (i2s_ctrl_t *const *p_api_ctrl*, i2s_dir_t const *dir*)

Stops SSI. Implements i2s_api_t::stop.

This function disables the transfer if the transfer interface is used, or sends a stop signal to stop writing data in the ISR if interrupt driven mode is used.

Return values

SSP_SUCCESS	I2S communication stop request issued.
SSP_ERR_ASSERTION	The pointer to p_ctrl null.
SSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

Stop is complete after an I2S_EVENT_IDLE interrupt.

◆ R_SSI_VersionGet()

ssp_err_t R_SSI_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ R_SSI_Write()

ssp_err_t R_SSI_Write (i2s_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint16_t const *bytes*)

Writes data buffer to SSI. Implements i2s_api_t::write.

This function resets the transfer if the transfer interface is used, or writes the length of data that fits in the FIFO then stores the remaining write buffer in the control block to be written in the ISR.

Write() cannot be called if another write(), read() or writeRead() operation is in progress. Write can be called when the SSI is idle, or after the I2S_EVENT_TX_EMPTY event.

Return values

SSP_SUCCESS	Write initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_src was null, or bytes requested was 0.
SSP_ERR_IN_USE	Another transfer is in progress, data was not written.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNDERFLOW	The transmit FIFO underflowed before it was reloaded.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset
- timer_api_t::start

If a transfer instance is provided for write, reset the transfer. Otherwise load the transmit FIFO.

Make sure transmission is enabled.

◆ R_SSI_WriteRead()

ssp_err_t R_SSI_WriteRead (i2s_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint16_t const *bytes*)

Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead.

This function calls R_SSI_Write and R_SSI_Read.

writeRead() cannot be called if another write(), read() or writeRead() operation is in progress. writeRead() can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

SSP_SUCCESS	Write and read initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl, p_src, or p_dest was null, or bytes requested was 0.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNDERFLOW	The transmit FIFO underflowed before it was reloaded.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset
- timer_api_t::start

If a transfer instance is provided for read, reset the transfer.

If a transfer instance is provided for write, reset the transfer.

Make sure transmission is enabled.

ssi_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SSI

#include <r_ssi.h>

Data Fields

void(*	p_callback)(i2s_callback_args_t *p_args)
void const *	p_context
R_SSI0_Type *	p_reg Pointer to SSI register base address.

timer_instance_t const *	p_timer Timer used to generate audio clock.
transfer_instance_t const *	p_transfer_tx Transfer used for hardware acceleration during write.
transfer_instance_t const *	p_transfer_rx Transfer used for hardware acceleration during read.
uint32_t const *	p_tx_src
uint32_t	tx_src_bytes
uint32_t *	p_rx_dest
uint32_t	rx_dest_bytes
uint32_t	sampling_freq_hz Sampling frequency in Hertz.
uint8_t	channel Channel number.
uint8_t	fifo_access_bytes Byte access to FIFO.
bool	tx_in_use True if a transmission is in progress.
bool	rx_in_use True if a reception is in progress.
bool	zeros_written True if zeros have been transmitted.
IRQn_Type	txi_irq

	Transmit IRQ number.
IRQn_Type	rxi_irq Receive IRQ number.
IRQn_Type	int_irq Idle/Error IRQ number.
uint32_t	open Whether or not this control block is initialized.

Detailed Description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when i2s_api_t::open is called.

Field Documentation

◆ p_callback

void(* ssi_instance_ctrl_t::p_callback) (i2s_callback_args_t *p_args)
Callback provided when an I2S ISR occurs. NULL indicates no CPU interrupt.

◆ p_context

void const* ssi_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

◆ p_rx_dest

uint32_t* ssi_instance_ctrl_t::p_rx_dest
Destination buffer pointer used to fill from hardware FIFO in receive ISR.

◆ p_tx_src

uint32_t const* ssi_instance_ctrl_t::p_tx_src
Source buffer pointer used to fill hardware FIFO from transmit ISR.

◆ rx_dest_bytes

uint32_t ssi_instance_ctrl_t::rx_dest_bytes
Size of destination buffer used to fill from hardware FIFO in receive ISR.

◆ tx_src_bytes

uint32_t ssi_instance_ctrl_t::tx_src_bytes
Size of source buffer used to fill hardware FIFO from transmit ISR.

The documentation for this struct was generated from the following file:

- r_ssi.h

i2s_on_ssi_cfg_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SSI

#include <r_ssi.h>

Data Fields

ssi_audio_clock_t	audio_clock Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL.

Detailed Description

SSI configuration extension. This extension is optional.

The documentation for this struct was generated from the following file:

- r_ssi.h

5.1.5.58 WDT

Renesas Synergy Software Package Reference » HAL Layer

Driver for the Watchdog Timer (WDT). More...

Data Structures

struct	wdt_instance_ctrl_t

Functions

ssp_err_t	R_WDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg) Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements wdt_api_t::open. More...
ssp_err_t	R_WDT_CfgGet (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t *const p_cfg) Read the configuration of the WDT in both register-start and auto-start modes. Implements wdt_api_t::cfgGet. More...
ssp_err_t	R_WDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout) Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet. More...
ssp_err_t	R_WDT_Refresh (wdt_ctrl_t *const p_api_ctrl) Refresh the watchdog timer. Implements wdt_api_t::refresh. More...
ssp_err_t	R_WDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status) Read the WDT status flags. Implements wdt_api_t::statusGet. More...
ssp_err_t	R_WDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status) Clear the WDT status and error flags. Implements wdt_api_t::statusClear. More...
ssp_err_t	R_WDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count) Read the current count value of the WDT. Implements wdt_api_t::counterGet. More...
ssp_err_t	R_WDT_VersionGet (ssp_version_t *const p_data) Return WDT HAL driver version. Implements wdt_api_t::versionGet.

More...

Detailed Description

Driver for the Watchdog Timer (WDT).

Summary

This module supports the Watchdog Timer (WDT). It implements the WDT Interface. The WDT HAL APIs provide the ability to configure the operation of the WDT (when used in register start mode), refresh the watchdog, read the timer value and read and clear status flags.

Function Documentation

◆ R_WDT_CfgGet()

ssp_err_t R_WDT_CfgGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t *const *p_cfg*)

Read the configuration of the WDT in both register-start and auto-start modes. Implements wdt_api_t::cfgGet.

Return values

SSP_SUCCESS	WDT successfully configured.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function is reentrant.

Register-start mode.

Get timeout value from WDTCR register.

◆ R_WDT_CounterGet()

ssp_err_t R_WDT_CounterGet (wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count*)

Read the current count value of the WDT. Implements wdt_api_t::counterGet.

Return values

SSP_SUCCESS	WDT current count successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.
SSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function is reentrant.

Read the WDT status

Get WDT down counter value

◆ R_WDT_Open()

ssp_err_t R_WDT_Open (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg*)

Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements wdt_api_t::open.

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Return values

SSP_SUCCESS	WDT successfully configured.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.
SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- fmi_api_t::productFeatureGet

Note

This function is reentrant. In auto-start mode the only valid configuration option is for registering the callback for the NMI ISR if NMI output has been selected.

g_wdt_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures this data structure is not optimised away.

Eliminate toolchain warning when NMI output is not being used.

Check the expected start mode matches the OSF0 configuration.

Lock the IWDT Hardware Resource

Initialize global pointer to WDT for NMI callback use.

Configuration only valid when WDT operating in register-start mode.

Register-start mode.

Register callback with BSP NMI ISR.

Enable the WDT underflow/refresh error interrupt (will generate an NMI).

Start the timer by performing a refresh.

◆ R_WDT_Refresh()

ssp_err_t R_WDT_Refresh (wdt_ctrl_t *const *p_api_ctrl*)

Refresh the watchdog timer. Implements wdt_api_t::refresh.

In addition to refreshing the watchdog counter this function can be used, in register start mode to start the counter.

Return values

SSP_SUCCESS	WDT successfully refreshed.
SSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function is reentrant. This function only returns SSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run. This function must not be called before calling R_WDT_Open().

Refresh the WDT Down counter

◆ R_WDT_StatusClear()

ssp_err_t R_WDT_StatusClear (wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status*)

Clear the WDT status and error flags. Implements wdt_api_t::statusClear.

Return values

SSP_SUCCESS	WDT flag(s) successfully cleared.
SSP_ERR_ASSERTION	Null pointer as a parameter.
SSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function is reentrant.

Write zero to clear flags.

◆ R_WDT_StatusGet()

ssp_err_t R_WDT_StatusGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status*)

Read the WDT status flags. Implements wdt_api_t::statusGet.

Indicates both status and error conditions.

Return values

SSP_SUCCESS	WDT status successfully read.
SSP_ERR_ASSERTION	Null pointer as a parameter.
SSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function is reentrant. When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

Read the WDT status

Get the value of refresh or underflow error flag

◆ R_WDT_TimeoutGet()

ssp_err_t R_WDT_TimeoutGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

SSP_SUCCESS	WDT timeout value successfully read.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

Note

This function is reentrant. This function must not be called before calling R_WDT_Open().

Read the configuration of the WDT

Get the frequency of the clock supplying the WDT

◆ R_WDT_VersionGet()

ssp_err_t R_WDT_VersionGet (ssp_version_t *const *p_data*)
Return WDT HAL driver version. Implements wdt_api_t::versionGet. **Return values** SSP_SUCCESS — Version information successfully read. SSP_ERR_ASSERTION — Null pointer passed as a parameter ***Note*** *This function is reentrant.*

wdt_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » WDT

#include <r_wdt.h>

Data Fields

uint32_t	wdt_open
void const *	p_context
R_WDT_Type *	p_reg Pointer to register base address.
void(*	p_callback)(wdt_callback_args_t *p_args) Callback provided when a WDT NMI ISR occurs.

Detailed Description

WDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called.

Field Documentation

◆ p_context

void const* wdt_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

◆ wdt_open

uint32_t wdt_instance_ctrl_t::wdt_open
Indicates whether the open() API has been successfully called.

The documentation for this struct was generated from the following file:

- r_wdt.h

5.1.5.59 SCE Module

Renesas Synergy Software Package Reference » HAL Layer

Primitive cryptographic functions. More...

Modules

	SCE AES Primitive cryptographic functions.
	SCE HRK AES Primitive cryptographic functions.
	SCE ARC4 Primitive cryptographic functions.
	SCE DSA Primitive cryptographic functions.
	SCE HASH Primitive cryptographic functions.
	SCE_ECC Primitive cryptographic functions.

	SCE_KEY_INSTALLATION Primitive cryptographic functions.
	SCE_RSA Primitive cryptographic functions.
	SCE_TDES Primitive cryptographic functions.
	SCE_TRNG Primitive cryptographic functions.
	SCE_INTERFACE_GET Get Interface for Crypto HAL modules.

Functions

uint32_t	R_SCE_Open (crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg) SCE Initialization - Opens the module and initializes the SCE. More...
uint32_t	R_SCE_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_StatusGet (uint32_t *p_status) This function indicates if the SCE has been initialized or not. The status returned is as follows: CRYPTO_SCE_COMMON_MODULE_CLOSED = 0 when the module is closed. CRYPTO_SCE_COMMON_MODULE_OPENED = 1 when the module is opened. More...
uint32_t	R_SCE_Close (crypto_ctrl_t *const p_ctrl) Close SCE driver. Close R_SCE driver by setting driver status to CRYPTO_SCE_COMMON_MODULE_CLOSED. More...

Detailed Description

Primitive cryptographic functions.

Initializes the SCE module cryptographic operations.

Crypto API interface on SCE

Function Documentation

◆ R_SCE_Close()

uint32_t R_SCE_Close (crypto_ctrl_t *const *p_ctrl*)

Close SCE driver. Close R_SCE driver by setting driver status to CRYPTO_SCE_COMMON_MODULE_CLOSED.

Parameters

[in]	p_ctrl	control structure for SCE block

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl input parameter is NULL.

◆ R_SCE_Open()

uint32_t R_SCE_Open (crypto_ctrl_t *const *p_ctrl*, crypto_cfg_t const *const *p_cfg*)

SCE Initialization - Opens the module and initializes the SCE.

Parameters

[in,out]	p_ctrl	control structure for the SCE module
[in]	p_cfg	configuration structure for the SCE module

Returns an uint32_t integer indicating:

Return values

SF_CRYPTO_SUCCESS	if the initialization is successful.
SSP_ERR_CRYPTO_SCE_FAIL	if the initialization failed.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	if the SCE hardware resource is busy.
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN	if the module is already open.
SSP_ERR_ASSERTION	NULL input parameter(s).

Get the list of API interfaces available to interfaceGet API.

◆ R_SCE_StatusGet()

uint32_t R_SCE_StatusGet (uint32_t * *p_status*)

This function indicates if the SCE has been initialized or not. The status returned is as follows:
CRYPTO_SCE_COMMON_MODULE_CLOSED = 0 when the module is closed.
CRYPTO_SCE_COMMON_MODULE_OPENED = 1 when the module is opened.

Parameters

[in]	p_status	pointer to uint32_t. This memory location is updated with the SCE module initialization status.

Return values

SF_CRYPTO_SUCCESS	API returned Successfully.
SSP_ERR_ASSERTION	The parameter p_status is NULL.

◆ R_SCE_VersionGet()

uint32_t R_SCE_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Parameters

[out]	p_version	version info for the SCE implementation

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

SCE AES

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_AES_Open (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES Open function. More...
uint32_t	R_SCE_AES_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_AES_Close (aes_ctrl_t *const p_ctrl) AES Close function. More...
uint32_t	R_SCE_AES_128CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit CBC mode implementation for encrypt interface API. More...
uint32_t	R_SCE_AES_128CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t

	*p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_128CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit CTR mode implementation for encrypt interface API. More...
uint32_t	R_SCE_AES_128EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit ECB mode implementation for encrypt interface API. More...
uint32_t	R_SCE_AES_128EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit ECB mode implementation for decrypt interface API. More...
uint32_t	R_SCE_AES_128GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES GCM 128-bit Open function. More...
uint32_t	R_SCE_AES_128GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_128GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 128-bit GCM Authentication Tag data. More...

uint32_t	R_SCE_AES_128GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member. More...
uint32_t	R_SCE_AES_128GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_128GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_128GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_128XtsEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_128XtsDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t

	*p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 128-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_192CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 192-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data. More...
uint32_t	R_SCE_AES_192CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 192-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data. More...
uint32_t	R_SCE_AES_192CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 192-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_192EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 192-bit ECB mode implementation for encrypt interface API. More...

uint32_t	R_SCE_AES_192EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 192-bit ECB mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_192GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES GCM 192-bit Open function. More...
uint32_t	R_SCE_AES_192GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_192GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 192-bit GCM Authentication Tag data. More...
uint32_t	R_SCE_AES_192GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success. More...
uint32_t	R_SCE_AES_192GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_192GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl,

	const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_192GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_256CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...

uint32_t	R_SCE_AES_256EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit ECB mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES GCM 256-bit Open function. More...
uint32_t	R_SCE_AES_256GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 256-bit GCM Authentication Tag data. More...
uint32_t	R_SCE_AES_256GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success. More...

uint32_t	R_SCE_AES_256GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_256GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data. More...
uint32_t	R_SCE_AES_256XtsEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_AES_256XtsDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) AES 256-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...

uint32_t	R_SCE_AES_EImpl_CreateKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) AES Create Key function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_EncryptFinal (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t output_num_words, uint32_t *p_dest) AES Encrypt input Key Final function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_ZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) AES Zero padding encrypt function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_ZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) AES Zero padding decrypt function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_SetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) AES Set GCM Tag function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_GetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) AES Get GCM Tag function, which is not implemented. More...
uint32_t	R_SCE_AES_EImpl_AddAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source) AES Adding additional authentication data function, which is not implemented. More...

Variables

const aes_api_t	g_aes128cbc_on_sce
const aes_api_t	g_aes128ctr_on_sce

const aes_api_t	g_aes128ecb_on_sce
const aes_api_t	g_aes128gcm_on_sce
const aes_api_t	g_aes128xts_on_sce
const aes_api_t	g_aes192cbc_on_sce
const aes_api_t	g_aes192ctr_on_sce
const aes_api_t	g_aes192ecb_on_sce
const aes_api_t	g_aes192gcm_on_sce
const aes_api_t	g_aes256cbc_on_sce
const aes_api_t	g_aes256ctr_on_sce
const aes_api_t	g_aes256ecb_on_sce
const aes_api_t	g_aes256gcm_on_sce
const aes_api_t	g_aes256xts_on_sce

Detailed Description

Primitive cryptographic functions.

AES key generation, encryption and decryption functions for plain-text / raw keys.

AES encryption and decryption functions

AES 256-bit CBC mode implementation for encryption and decryption functions

AES 256-bit CTR mode implementation for encryption and decryption functions

AES 256-bit ECB mode implementation for encryption and decryption functions

AES 256-bit XTS mode implementation for encryption and decryption functions

Function Documentation

◆ R_SCE_AES_128CbcDecrypt()

uint32_t R_SCE_AES_128CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_128CbcEncrypt()

uint32_t R_SCE_AES_128CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit CBC mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_128CtrEncrypt()

uint32_t R_SCE_AES_128CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit CTR mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_128EcbDecrypt()

uint32_t R_SCE_AES_128EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit ECB mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_128EcbEncrypt()

uint32_t R_SCE_AES_128EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit ECB mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_128GcmDecrypt()

uint32_t R_SCE_AES_128GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_128GcmEncrypt()

uint32_t R_SCE_AES_128GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_128GcmGetGcmTag()

uint32_t R_SCE_AES_128GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 128-bit GCM Authentication Tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_128GcmOpen()

uint32_t R_SCE_AES_128GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES GCM 128-bit Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_AES_128GcmSetGcmTag()

uint32_t R_SCE_AES_128GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_128GcmZeroPaddingDecrypt()

uint32_t R_SCE_AES_128GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_128GcmZeroPaddingEncrypt()

uint32_t R_SCE_AES_128GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

*p_dest must have space to hold at least (num_bytes/16+1)*16 bytes of data.*
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
this function is not thread safe.

◆ R_SCE_AES_128XtsDecrypt()

uint32_t R_SCE_AES_128XtsDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_128XtsEncrypt()

uint32_t R_SCE_AES_128XtsEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 128-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_192CbcDecrypt()

uint32_t R_SCE_AES_192CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 192-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 24 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_192CbcEncrypt()

uint32_t R_SCE_AES_192CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 192-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 24 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_192CtrEncrypt()

uint32_t R_SCE_AES_192CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 192-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 24 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_192EcbDecrypt()

uint32_t R_SCE_AES_192EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 192-bit ECB mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 24 bytes of AES key data and
The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_192EcbEncrypt()

uint32_t R_SCE_AES_192EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 192-bit ECB mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 24 bytes of AES key data and
The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_192GcmDecrypt()

uint32_t R_SCE_AES_192GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_192GcmEncrypt()

uint32_t R_SCE_AES_192GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_192GcmGetGcmTag()

uint32_t R_SCE_AES_192GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 192-bit GCM Authentication Tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_192GcmOpen()

uint32_t R_SCE_AES_192GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES GCM 192-bit Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_AES_192GcmSetGcmTag()

uint32_t R_SCE_AES_192GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_192GcmZeroPaddingDecrypt()

uint32_t R_SCE_AES_192GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_192GcmZeroPaddingEncrypt()

uint32_t R_SCE_AES_192GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data
this function is not thread safe.

◆ R_SCE_AES_256CbcDecrypt()

uint32_t R_SCE_AES_256CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_256CbcEncrypt()

uint32_t R_SCE_AES_256CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_256CtrEncrypt()

uint32_t R_SCE_AES_256CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_256EcbDecrypt()

uint32_t R_SCE_AES_256EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Note

The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_256EcbEncrypt()

uint32_t R_SCE_AES_256EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit ECB mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Note

The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

◆ R_SCE_AES_256GcmDecrypt()

uint32_t R_SCE_AES_256GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_256GcmEncrypt()

uint32_t R_SCE_AES_256GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_256GcmGetGcmTag()

uint32_t R_SCE_AES_256GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 256-bit GCM Authentication Tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_256GcmOpen()

uint32_t R_SCE_AES_256GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES GCM 256-bit Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_AES_256GcmSetGcmTag()

uint32_t R_SCE_AES_256GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_AES_256GcmZeroPaddingDecrypt()

uint32_t R_SCE_AES_256GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

◆ R_SCE_AES_256GcmZeroPaddingEncrypt()

uint32_t R_SCE_AES_256GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
this function is not thread safe.

◆ R_SCE_AES_256XtsDecrypt()

uint32_t R_SCE_AES_256XtsDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_256XtsEncrypt()

uint32_t R_SCE_AES_256XtsEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES 256-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_AES_Close()

uint32_t R_SCE_AES_Close (aes_ctrl_t *const *p_ctrl*)

AES Close function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

◆ R_SCE_AES_EImpl_AddAdditionalAuthenticationData()

uint32_t R_SCE_AES_EImpl_AddAdditionalAuthenticationData (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*)

AES Adding additional authentication data function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	p_key	pointer to the AES plain-text key
[in]	p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	num_words	data buffer size in words. Each work is 4-bytes and multiples of 4.
[in]	p_source	input data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_CreateKey()

uint32_t R_SCE_AES_EImpl_CreateKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

AES Create Key function, which is not implemented.

SCE/AES implementation of AES API.

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_EncryptFinal()

uint32_t R_SCE_AES_EImpl_EncryptFinal (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *input_num_words*, uint32_t * *p_source*, uint32_t *output_num_words*, uint32_t * *p_dest*)

AES Encrypt input Key Final function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	p_key	pointer to the AES plain-text key
[in]	p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	input_num_words	input data buffer size in words. Each word is 4-bytes and multiples of 4.
[in]	p_source	input data buffer
[out]	output_num_words	output data buffer size in words. Each word is 4-bytes and multiples of 4.
[in,out]	p_dest	output data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_GetGcmTag()

uint32_t R_SCE_AES_EImpl_GetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

AES Get GCM Tag function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	num_words	data buffer size in words. Each work is 4-bytes and multiples of 4.
[in,out]	p_dest	output data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_SetGcmTag()

uint32_t R_SCE_AES_EImpl_SetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

AES Set GCM Tag function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	num_words	data buffer size in words. Each work is 4-bytes and multiples of 4.
[in]	p_source	input data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_ZeroPaddingDecrypt()

uint32_t R_SCE_AES_EImpl_ZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES Zero padding decrypt function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	p_key	pointer to the AES plain-text key
[in]	p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	num_bytes	data buffer size in bytes.
[in]	p_source	input data buffer
[in,out]	p_dest	output data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_EImpl_ZeroPaddingEncrypt()

uint32_t R_SCE_AES_EImpl_ZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

AES Zero padding encrypt function, which is not implemented.

Parameters

[in]	p_ctrl	pointer to the control structure
[in]	p_key	pointer to the AES plain-text key
[in]	p_iv	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
[in]	num_bytes	data buffer size in bytes.
[in]	p_source	input data buffer
[in,out]	p_dest	output data buffer

Return values

SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_AES_Open()

uint32_t R_SCE_AES_Open (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_AES_VersionGet()

uint32_t R_SCE_AES_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_aes128cbc_on_sce

const aes_api_t g_aes128cbc_on_sce

```
=
{
    .open                           = R_SCE_AES_Open,
    .encrypt                        = R_SCE_AES_128CbcEncrypt,
    .decrypt                        = R_SCE_AES_128CbcDecrypt,
    .close                          = R_SCE_AES_Close,
    .versionGet                     = R_SCE_AES_VersionGet,
    .createKey                     = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                  = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt            = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt            = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                     = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                     = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 128-bit CBC mode implementation

◆ g_aes128ctr_on_sce

const aes_api_t g_aes128ctr_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_128CtrEncrypt,
    .decrypt                            = R_SCE_AES_128CtrEncrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 128-bit CTR mode implementation

◆ g_aes128ecb_on_sce

const aes_api_t g_aes128ecb_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_128EcbEncrypt,
    .decrypt                            = R_SCE_AES_128EcbDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 128-bit ECB mode implementation

◆ g_aes128gcm_on_sce

const aes_api_t g_aes128gcm_on_sce

```
=
{
    .open                               = R_SCE_AES_128GcmOpen,
    .encrypt                            = R_SCE_AES_128GcmEncrypt,
    .decrypt                            = R_SCE_AES_128GcmDecrypt,
    .getGcmTag                         = R_SCE_AES_128GcmGetGcmTag,
    .setGcmTag                         = R_SCE_AES_128GcmSetGcmTag,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                = R_SCE_AES_128GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_128GcmZeroPaddingDecrypt,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData
}
```

AES 128-bit GCM mode implementation

◆ g_aes128xts_on_sce

const aes_api_t g_aes128xts_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_128XtsEncrypt,
    .decrypt                            = R_SCE_AES_128XtsDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 128-bit CCM mode implementation

◆ g_aes192cbc_on_sce

const aes_api_t g_aes192cbc_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_192CbcEncrypt,
    .decrypt                            = R_SCE_AES_192CbcDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 192-bit CBC mode implementation

◆ g_aes192ctr_on_sce

const aes_api_t g_aes192ctr_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_192CtrEncrypt,
    .decrypt                            = R_SCE_AES_192CtrEncrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 192-bit CTR mode implementation

◆ g_aes192ecb_on_sce

const aes_api_t g_aes192ecb_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_192EcbEncrypt,
    .decrypt                            = R_SCE_AES_192EcbDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 192-bit ECB mode implementation

◆ g_aes192gcm_on_sce

const aes_api_t g_aes192gcm_on_sce

```
=
{
    .open                               = R_SCE_AES_192GcmOpen,
    .encrypt                            = R_SCE_AES_192GcmEncrypt,
    .decrypt                            = R_SCE_AES_192GcmDecrypt,
    .getGcmTag                         = R_SCE_AES_192GcmGetGcmTag,
    .setGcmTag                         = R_SCE_AES_192GcmSetGcmTag,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                = R_SCE_AES_192GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_192GcmZeroPaddingDecrypt,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
}
```

AES 192-bit GCM mode implementation

◆ g_aes256cbc_on_sce

const aes_api_t g_aes256cbc_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_256CbcEncrypt,
    .decrypt                            = R_SCE_AES_256CbcDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 256-bit CBC mode implementation

◆ g_aes256ctr_on_sce

const aes_api_t g_aes256ctr_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_256CtrEncrypt,
    .decrypt                            = R_SCE_AES_256CtrEncrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 256-bit CTR mode implementation

◆ g_aes256ecb_on_sce

const aes_api_t g_aes256ecb_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_256EcbEncrypt,
    .decrypt                            = R_SCE_AES_256EcbDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 256-bit ECB mode implementation

◆ g_aes256gcm_on_sce

const aes_api_t g_aes256gcm_on_sce

```
=
{
    .open                               = R_SCE_AES_256GcmOpen,
    .encrypt                            = R_SCE_AES_256GcmEncrypt,
    .decrypt                            = R_SCE_AES_256GcmDecrypt,
    .getGcmTag                         = R_SCE_AES_256GcmGetGcmTag,
    .setGcmTag                         = R_SCE_AES_256GcmSetGcmTag,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                = R_SCE_AES_256GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_256GcmZeroPaddingDecrypt,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
}
```

AES 256-bit GCM mode implementation

◆ g_aes256xts_on_sce

const aes_api_t g_aes256xts_on_sce

```
=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_256XtsEncrypt,
    .decrypt                            = R_SCE_AES_256XtsDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

AES 256-bit XTS mode implementation

SCE HRK AES

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_HRK_AES_Open (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES Open function. More...
uint32_t	R_SCE_HRK_AES_Close (aes_ctrl_t *const p_ctrl) AES Close function. More...

uint32_t	R_SCE_HRK_AES_128CreateEncryptedKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Encrypted Wrapped key creation for AES 128-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data. More...
uint32_t	R_SCE_HRK_AES_192CreateEncryptedKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Encrypted Wrapped key creation for AES 192-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data. More...
uint32_t	R_SCE_HRK_AES_256CreateEncryptedKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Encrypted Wrapped key creation for AES 256-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data. More...
uint32_t	R_SCE_HRK_AES_128CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit CBC mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_128CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit CBC mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_128CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit CTR mode implementation for encrypt interface API. More...

uint32_t	R_SCE_HRK_AES_128EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit ECB mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_128EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit ECB mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_128GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES Open function. More...
uint32_t	R_SCE_HRK_AES_128GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_128GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_128GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 128-bit GCM authentication tag data. More...
uint32_t	R_SCE_HRK_AES_128GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) Sets the expected authentication tag value for the AES-GCM-HRK

	Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success. More...
uint32_t	R_SCE_HRK_AES_128GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_128GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_128XtsCreateEncryptedKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Encrypted Wrapped key creation for AES 1286-bit XTS mode implementation. More...
uint32_t	R_SCE_HRK_AES_128XtsEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit XTS mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_128XtsDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted Key AES 128-bit XTS mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_192CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 192-bit CBC mode implementation for encrypt

	interface API. More...
uint32_t	R_SCE_HRK_AES_192CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 192-bit CBC mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_192CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 192-bit CTR mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_192EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 192-bit ECB mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_192EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 192-bit ECB mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_192GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES Open function. More...
uint32_t	R_SCE_HRK_AES_192GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 192-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_192GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 192-bit GCM authentication tag data. More...

uint32_t	R_SCE_HRK_AES_192GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source) Sets the expected authentication tag value for the AES-GCM-HRK Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success. More...
uint32_t	R_SCE_HRK_AES_192GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_192GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 192-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_192GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes bytes of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_256CbcEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted AES 256-bit CBC mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_256CbcDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t

	*p_source, uint32_t *p_dest) Encrypted AES 256-bit ECB mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_256CtrEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 256-bit CTR mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_256EcbEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 256-bit ECB mode implementation for encrypt interface API. More...
uint32_t	R_SCE_HRK_AES_256EcbDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 256-bit ECB mode implementation for decrypt interface API. More...
uint32_t	R_SCE_HRK_AES_256GcmOpen (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg) AES Open function. More...
uint32_t	R_SCE_HRK_AES_256GcmEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_256GcmGetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest) Get AES 256-bit GCM authentication tag data. More...
uint32_t	R_SCE_HRK_AES_256GcmSetGcmTag (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)

	Sets the expected authentication tag value for the AES-GCM-HRK Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success. More...
uint32_t	R_SCE_HRK_AES_256GcmDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...
uint32_t	R_SCE_HRK_AES_256GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Encrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_256GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest) Decrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block. More...
uint32_t	R_SCE_HRK_AES_256XtsCreateEncryptedKey (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key) Encrypted Wrapped key creation for AES 256-bit XTS mode implementation. More...
uint32_t	R_SCE_HRK_AES_256XtsEncrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 256-bit XTS mode implementation for encrypt interface API. More...

uint32_t	R_SCE_HRK_AES_256XtsDecrypt (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypted key AES 256-bit XTS mode implementation for decrypt interface API. More...

Variables

const aes_api_t	g_aes128ecb_on_sceHrk
const aes_api_t	g_aes192gcm_on_sceHrk

Detailed Description

Primitive cryptographic functions.

AES key generation, encryption and decryption functions for wrapped keys.

SCE HRK Encrypted Key AES Cipher implementation functions

Function Documentation

◆ R_SCE_HRK_AES_128CbcDecrypt()

uint32_t R_SCE_HRK_AES_128CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit CBC mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().

◆ R_SCE_HRK_AES_128CbcEncrypt()

uint32_t R_SCE_HRK_AES_128CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit CBC mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().

◆ R_SCE_HRK_AES_128CreateEncryptedKey()

uint32_t R_SCE_HRK_AES_128CreateEncryptedKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

Encrypted Wrapped key creation for AES 128-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_ctrl parameter is not used

◆ R_SCE_HRK_AES_128CtrEncrypt()

uint32_t R_SCE_HRK_AES_128CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit CTR mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using a wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().

◆ R_SCE_HRK_AES_128EcbDecrypt()

uint32_t R_SCE_HRK_AES_128EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit ECB mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_128EcbEncrypt()

uint32_t R_SCE_HRK_AES_128EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit ECB mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_128GcmDecrypt()

uint32_t R_SCE_HRK_AES_128GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_128GcmEncrypt()

uint32_t R_SCE_HRK_AES_128GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_128GcmGetGcmTag()

uint32_t R_SCE_HRK_AES_128GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 128-bit GCM authentication tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_128GcmOpen()

uint32_t R_SCE_HRK_AES_128GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_HRK_AES_128GcmSetGcmTag()

uint32_t R_SCE_HRK_AES_128GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM-HRK Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_128GcmZeroPaddingDecrypt()

uint32_t R_SCE_HRK_AES_128GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_128GcmZeroPaddingEncrypt()

uint32_t R_SCE_HRK_AES_128GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_128XtsCreateEncryptedKey()

uint32_t R_SCE_HRK_AES_128XtsCreateEncryptedKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

Encrypted Wrapped key creation for AES 1286-bit XTS mode implementation.

Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4 and must be less than 128M

◆ R_SCE_HRK_AES_128XtsDecrypt()

uint32_t R_SCE_HRK_AES_128XtsDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit XTS mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using a wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4 and must be less than 128M
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncryptedKey().

Verify the upper-limit of num_words

HW_SCE_AES_128XtsDecryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

◆ R_SCE_HRK_AES_128XtsEncrypt()

uint32_t R_SCE_HRK_AES_128XtsEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted Key AES 128-bit XTS mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4 and must be less than 128M
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128XtsCreateEncryptedKey().

Verify the upper-limit of num_words

HW_SCE_AES_128XtsEncryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

◆ R_SCE_HRK_AES_192CbcDecrypt()

uint32_t R_SCE_HRK_AES_192CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 192-bit CBC mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncryptedKey().

◆ R_SCE_HRK_AES_192CbcEncrypt()

uint32_t R_SCE_HRK_AES_192CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 192-bit CBC mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 192-bit AES key generated using R_SCE_HRK_AES_192CreateEncryptedKey().

◆ R_SCE_HRK_AES_192CreateEncryptedKey()

uint32_t R_SCE_HRK_AES_192CreateEncryptedKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

Encrypted Wrapped key creation for AES 192-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_ctrl parameter is not used

◆ R_SCE_HRK_AES_192CtrEncrypt()

uint32_t R_SCE_HRK_AES_192CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 192-bit CTR mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncryptedKey().

◆ R_SCE_HRK_AES_192EcbDecrypt()

uint32_t R_SCE_HRK_AES_192EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 192-bit ECB mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_192EcbEncrypt()

uint32_t R_SCE_HRK_AES_192EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 192-bit ECB mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_192GcmDecrypt()

uint32_t R_SCE_HRK_AES_192GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_192GcmEncrypt()

uint32_t R_SCE_HRK_AES_192GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 192-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_192GcmGetGcmTag()

uint32_t R_SCE_HRK_AES_192GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 192-bit GCM authentication tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_192GcmOpen()

uint32_t R_SCE_HRK_AES_192GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_HRK_AES_192GcmSetGcmTag()

uint32_t R_SCE_HRK_AES_192GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM-HRK Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_192GcmZeroPaddingDecrypt()

uint32_t R_SCE_HRK_AES_192GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes bytes of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_192GcmZeroPaddingEncrypt()

uint32_t R_SCE_HRK_AES_192GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 192-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_256CbcDecrypt()

uint32_t R_SCE_HRK_AES_256CbcDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted AES 256-bit ECB mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncryptedKey().

◆ R_SCE_HRK_AES_256CbcEncrypt()

uint32_t R_SCE_HRK_AES_256CbcEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted AES 256-bit CBC mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncryptedKey().

◆ R_SCE_HRK_AES_256CreateEncryptedKey()

uint32_t R_SCE_HRK_AES_256CreateEncryptedKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

Encrypted Wrapped key creation for AES 256-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_ctrl parameter is not used

◆ R_SCE_HRK_AES_256CtrEncrypt()

uint32_t R_SCE_HRK_AES_256CtrEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 256-bit CTR mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncryptedKey().

◆ R_SCE_HRK_AES_256EcbDecrypt()

uint32_t R_SCE_HRK_AES_256EcbDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 256-bit ECB mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_256EcbEncrypt()

uint32_t R_SCE_HRK_AES_256EcbEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 256-bit ECB mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncryptedKey().
The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

◆ R_SCE_HRK_AES_256GcmDecrypt()

uint32_t R_SCE_HRK_AES_256GcmDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_256GcmEncrypt()

uint32_t R_SCE_HRK_AES_256GcmEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.

◆ R_SCE_HRK_AES_256GcmGetGcmTag()

uint32_t R_SCE_HRK_AES_256GcmGetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_dest*)

Get AES 256-bit GCM authentication tag data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_256GcmOpen()

uint32_t R_SCE_HRK_AES_256GcmOpen (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_HRK_AES_256GcmSetGcmTag()

uint32_t R_SCE_HRK_AES_256GcmSetGcmTag (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_source*)

Sets the expected authentication tag value for the AES-GCM-HRK Decryption related functions. Copies the provided tag value to the corresponding internal control structure member return success.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

◆ R_SCE_HRK_AES_256GcmZeroPaddingDecrypt()

uint32_t R_SCE_HRK_AES_256GcmZeroPaddingDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_256GcmZeroPaddingEncrypt()

uint32_t R_SCE_HRK_AES_256GcmZeroPaddingEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_bytes*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data and padded with 0's to make 16-byte block.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed
SSP_ERR_ASSERTION	An input parameter is NULL.
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least num_bytes bytes of data.
The p_key buffer must contain 16 bytes of AES key data and
the p_iv buffer must have at least 16 bytes of random data.
p_dest to be padded with '0's to make 16-byte block.

◆ R_SCE_HRK_AES_256XtsCreateEncryptedKey()

uint32_t R_SCE_HRK_AES_256XtsCreateEncryptedKey (aes_ctrl_t *const *p_ctrl*, uint32_t *num_words*, uint32_t * *p_key*)

Encrypted Wrapped key creation for AES 256-bit XTS mode implementation.

Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4

◆ R_SCE_HRK_AES_256XtsDecrypt()

uint32_t R_SCE_HRK_AES_256XtsDecrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 256-bit XTS mode implementation for decrypt interface API.

Decrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4 and must be less than 128M
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256XtsCreateEncryptedKey().

Verify the upper-limit of num_words

HW_SCE_AES_256XtsDecryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

◆ R_SCE_HRK_AES_256XtsEncrypt()

uint32_t R_SCE_HRK_AES_256XtsEncrypt (aes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypted key AES 256-bit XTS mode implementation for encrypt interface API.

Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size
SSP_ERR_ASSERTION	An input parameter is NULL.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

num_words must be a multiple of 4 and must be less than 128M
p_dest must have space to hold at least num_words words of data.
The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256XtsCreateEncryptedKey().

Verify the upper-limit of num_words

HW_SCE_AES_256XtsEncryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

◆ R_SCE_HRK_AES_Close()

uint32_t R_SCE_HRK_AES_Close (aes_ctrl_t *const *p_ctrl*)

AES Close function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

◆ R_SCE_HRK_AES_Open()

uint32_t R_SCE_HRK_AES_Open (aes_ctrl_t *const *p_ctrl*, aes_cfg_t const *const *p_cfg*)

AES Open function.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

Variable Documentation

◆ g_aes128ecb_on_sceHrk

const aes_api_t g_aes128ecb_on_sceHrk

```
=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_128EcbEncrypt,
    .decrypt                            = R_SCE_HRK_AES_128EcbDecrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_HRK_AES_128CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

HRK Supported global structure definitions

◆ g_aes192gcm_on_sceHrk

const aes_api_t g_aes192gcm_on_sceHrk

```
=
{
    .open       = R_SCE_HRK_AES_192GcmOpen,
    .createKey = R_SCE_HRK_AES_192CreateEncryptedKey,
    .encrypt    = R_SCE_HRK_AES_192GcmEncrypt,
    .decrypt    = R_SCE_HRK_AES_192GcmDecrypt,
    .getGcmTag = R_SCE_HRK_AES_192GcmGetGcmTag,
    .setGcmTag = R_SCE_HRK_AES_192GcmSetGcmTag,
    .close      = R_SCE_HRK_AES_Close,
    .versionGet = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt = R_SCE_HRK_AES_192GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt = R_SCE_HRK_AES_192GcmZeroPaddingDecrypt
}
```

AES 192-bit GCM HRK mode implementation

SCE ARC4

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_ARC4_Open (arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)
uint32_t	R_SCE_ARC4_Close (arc4_ctrl_t *const p_ctrl)
uint32_t	R_SCE_ARC4_Process (arc4_ctrl_t *const p_ctrl, uint32_t num_bytes, uint8_t *p_source, uint8_t *p_dest) ARC4 Encrypt or decrypt source data and output result to destination buffer. More...
uint32_t	R_SCE_ARC4_VersionGet (ssp_version_t *const p_version)

	Sets driver version based on compile time macros. More...
uint32_t	R_SCE_ARC4_KeySet (arc4_ctrl_t *const p_ctrl, uint32_t length, uint8_t const *p_key) Sets user provided for use with subsequent encryptions. More...

Detailed Description

Primitive cryptographic functions.

ARC4 encryption and decryption functions

Function Documentation

◆ R_SCE_ARC4_Close()

uint32_t R_SCE_ARC4_Close (arc4_ctrl_t *const *p_ctrl*)

ARC4 Finalization

Return values

SF_CRYPTO_SUCCESS	successful

◆ R_SCE_ARC4_KeySet()

uint32_t R_SCE_ARC4_KeySet (arc4_ctrl_t *const *p_ctrl*, uint32_t *length*, uint8_t const * *p_key*)

Sets user provided for use with subsequent encryptions.

Return values

SF_CRYPTO_SUCCESS	KeySet successful
SSP_ERR_NOT_OPEN	Module not opened.
SSP_ERR_ASSERTION	One of input parameter is NULL.

◆ R_SCE_ARC4_Open()

uint32_t R_SCE_ARC4_Open (arc4_ctrl_t *const *p_ctrl*, arc4_cfg_t const *const *p_cfg*)

ARC4 Initialization

Return values

SF_CRYPTO_SUCCESS	successful.
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN	ARC4 module is already in open state and is usable for the given p_ctrl parameter.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_ARC4_Process()

uint32_t R_SCE_ARC4_Process (arc4_ctrl_t *const *p_ctrl*, uint32_t *num_bytes*, uint8_t * *p_source*, uint8_t * *p_dest*)

ARC4 Encrypt or decrypt source data and output result to destination buffer.

Return values

SSP_ERR_CRYPTO_INVALID_STATE	Invalid state, ensure that key data is configured either using the open() or using the keyset() function.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid num_bytes passed. Should be a multiple of 16.
SF_CRYPTO_SUCCESS	successful.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	An internal error has occurred.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware is busy, unable to encrypt at this time.

Encrypt or decrypt input data using the previously configured key data

◆ R_SCE_ARC4_VersionGet()

uint32_t R_SCE_ARC4_VersionGet (ssp_version_t *const *p_version*)
Sets driver version based on compile time macros. **Return values** SSP_SUCCESS — Successful close. SSP_ERR_ASSERTION — The parameter p_version is NULL.

SCE DSA

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_DSA_Open (dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)
uint32_t	R_SCE_DSA_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_DSA_Close (dsa_ctrl_t *const p_ctrl)
uint32_t	R_SCE_DSA_1024_160SignatureVerify (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature verification using (1024-bit,160-bit) DSA public key. More...
uint32_t	R_SCE_DSA_1024_160SignatureGenerate (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using (1024-bit,160-bit) DSA private key. More...
uint32_t	R_SCE_DSA_1024_160HashSignatureVerify (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash)

	Signature verification using (1024-bit,160-bit) DSA public key. More...
uint32_t	R_SCE_DSA_1024_160HashSignatureGenerate (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using (1024-bit,160-bit) DSA private key. More...
uint32_t	R_SCE_DSA_2048_224SignatureVerify (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature verification using (2048-bit,224-bit) DSA public key. More...
uint32_t	R_SCE_DSA_2048_224SignatureGenerate (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using (2048-bit,224-bit) DSA private key. More...
uint32_t	R_SCE_DSA_2048_224HashSignatureVerify (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature verification using (2048-bit,224-bit) DSA public key. More...
uint32_t	R_SCE_DSA_2048_224HashSignatureGenerate (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using (2048-bit,224-bit) DSA private key. More...
uint32_t	R_SCE_DSA_2048_256SignatureVerify (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature verification using (2048-bit,256-bit) DSA public key. More...
uint32_t	R_SCE_DSA_2048_256SignatureGenerate (const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)

	Signature generation using (2048-bit,256-bit) DSA private key. More...
uint32_t	R_SCE_DSA_2048_256HashSignatureVerify (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature verification using (2048-bit,256-bit) DSA public key. More...
uint32_t	R_SCE_DSA_2048_256HashSignatureGenerate (dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using (2048-bit,256-bit) DSA private key. More...

Variables

const dsa_api_t	g_dsa1024_160_on_sce
const dsa_api_t	g_dsa2048_224_on_sce
const dsa_api_t	g_dsa2048_256_on_sce

Detailed Description

Primitive cryptographic functions.

Primitive cryptographic functions (L=2048,N=256) DSA.

DSA signature generation and verification functions

DSA signature generation and verification functions (L=1024,N=160) DSA

DSA signature generation and verification functions (L=2048,N=224) DSA

Function Documentation

◆ R_SCE_DSA_1024_160HashSignatureGenerate()

uint32_t R_SCE_DSA_1024_160HashSignatureGenerate (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (1024-bit,160-bit) DSA private key.

Sign imaxcnt words of input data from buffer p_source using the (L=1024,N=160)-bit DSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast 2*imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The p_key buffer must contain a valid DSA private key data and p_domain should contain DSA domain parameters in the order (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

◆ R_SCE_DSA_1024_160HashSignatureVerify()

uint32_t R_SCE_DSA_1024_160HashSignatureVerify (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (1024-bit,160-bit) DSA public key.

Verify DSA signature data from buffer p_signature of length 2*imaxcnt words using (L=1024,N=160) DSA public key from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Incorrect signature
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 32 words of DSA public key data
The p_domain buffer must contain (20+128+128) bytes of data in the format (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

◆ R_SCE_DSA_1024_160SignatureGenerate()

uint32_t R_SCE_DSA_1024_160SignatureGenerate (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (1024-bit,160-bit) DSA private key.

Sign imaxcnt words of input data from buffer p_source using the (L=1024,N=160)-bit DSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast 2*imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The p_key buffer must contain a valid DSA private key data and p_domain should contain DSA domain parameters in the order (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

◆ R_SCE_DSA_1024_160SignatureVerify()

uint32_t R_SCE_DSA_1024_160SignatureVerify (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (1024-bit,160-bit) DSA public key.

Verify DSA signature data from buffer p_signature of length 2*imaxcnt words using (L=1024,N=160) DSA public key from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Incorrect signature
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 32 words of DSA public key data
The p_domain buffer must contain (20+128+128) bytes of data in the format (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

◆ R_SCE_DSA_2048_224HashSignatureGenerate()

uint32_t R_SCE_DSA_2048_224HashSignatureGenerate (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (2048-bit,224-bit) DSA private key.

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=224)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The key buffer must contain 7 words of DSA private key data
p_domain must contain valid DSA domain parameters

◆ R_SCE_DSA_2048_224HashSignatureVerify()

uint32_t R_SCE_DSA_2048_224HashSignatureVerify (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (2048-bit,224-bit) DSA public key.

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Signature verification failed.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 64 words of DSA public key data
The p_domain buffer must contain (28+256+256) bytes of data in the format (Q || P || G) where Q is 7 words, P is 64 words and G is 64 words.

◆ R_SCE_DSA_2048_224SignatureGenerate()

uint32_t R_SCE_DSA_2048_224SignatureGenerate (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (2048-bit,224-bit) DSA private key.

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=224)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The key buffer must contain 7 words of DSA private key data
p_domain must contain valid DSA domain parameters

◆ R_SCE_DSA_2048_224SignatureVerify()

uint32_t R_SCE_DSA_2048_224SignatureVerify (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (2048-bit,224-bit) DSA public key.

Verify DSA signature data from buffer p_signature of length (2 * 'imaxcnt' words) using (L=2048,N=224) DSA public key. from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Signature verification failed.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 256 bytes of DSA public key data.
The p_domain buffer must contain (28+256+256) bytes of data in the format (Q || P || G).

◆ R_SCE_DSA_2048_256HashSignatureGenerate()

uint32_t R_SCE_DSA_2048_256HashSignatureGenerate (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (2048-bit,256-bit) DSA private key.

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=256)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The key buffer must contain 8 words of DSA private key data

◆ R_SCE_DSA_2048_256HashSignatureVerify()

uint32_t R_SCE_DSA_2048_256HashSignatureVerify (dsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (2048-bit,256-bit) DSA public key.

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Return values

SF_CRYPTO_SUCCESS	Call successful.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Signature verification failed.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 256 bytes of DSA public key data
The p_domain buffer must contain (32+256+256) bytes of data in the format (Q || P || G).

◆ R_SCE_DSA_2048_256SignatureGenerate()

uint32_t R_SCE_DSA_2048_256SignatureGenerate (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using (2048-bit,256-bit) DSA private key.

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=256)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

*p_dest must have space to hold at least 2*imaxcnt words of data.*
The key buffer must contain 8 words of DSA private key data

◆ R_SCE_DSA_2048_256SignatureVerify()

uint32_t R_SCE_DSA_2048_256SignatureVerify (const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature verification using (2048-bit,256-bit) DSA public key.

Verify DSA signature data from buffer p_signature of length (2 * 'imaxcnt' words) using (L=2048,N=224) DSA public key. from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Return values

SF_CRYPTO_SUCCESS	Call successful.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Signature verification failed.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE DSA module must have been initialized by calling dsa_api_t::open

Note

The p_key buffer must contain 256 bytes of DSA public key data
The p_domain buffer must contain (32+256+256) bytes of data in the format (Q || P || G).

◆ R_SCE_DSA_Close()

uint32_t R_SCE_DSA_Close (dsa_ctrl_t *const *p_ctrl*)

Close DSA driver

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

◆ R_SCE_DSA_Open()

uint32_t R_SCE_DSA_Open (dsa_ctrl_t *const *p_ctrl*, dsa_cfg_t const *const *p_cfg*)

DSA Initialization

Return values

SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_DSA_VersionGet()

uint32_t R_SCE_DSA_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_dsa1024_160_on_sce

```
const dsa_api_t g_dsa1024_160_on_sce
=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_1024_160SignatureGenerate,
    .verify    = R_SCE_DSA_1024_160SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_1024_160HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_1024_160HashSignatureVerify
}
```

SCE/DSA implementation of DSA API.

◆ g_dsa2048_224_on_sce

```
const dsa_api_t g_dsa2048_224_on_sce
=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_2048_224SignatureGenerate,
    .verify    = R_SCE_DSA_2048_224SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_2048_224HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_2048_224HashSignatureVerify,
}
```

SCE/DSA implementation of DSA API.

◆ g_dsa2048_256_on_sce

const dsa_api_t g_dsa2048_256_on_sce

```
=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_2048_256SignatureGenerate,
    .verify    = R_SCE_DSA_2048_256SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_2048_256HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_2048_256HashSignatureVerify
}
```

SCE/DSA implementation of DSA API.

SCE HASH

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_HASH_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_MD5_Open (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg) SCE MD5 open function. More...
uint32_t	R_SCE_MD5_Close (hash_ctrl_t *const p_ctrl) SCE MD5 Close function. More...
uint32_t	R_SCE_MD5_UpdateHash (const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest)

	MD5 Update Hash function. More...
uint32_t	R_SCE_MD5_HashUpdate (hash_ctrl_t *const p_ctrl, const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest) MD5 HashUpdate Function. More...
uint32_t	R_SCE_SHA1_Open (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg) SCE SHA1 open function using the SCE block. More...
uint32_t	R_SCE_SHA1_Close (hash_ctrl_t *const p_ctrl) SCE SHA1 Close function. More...
uint32_t	R_SCE_SHA1_UpdateHash (const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest) Computes the SHA1 message digest of the input message. More...
uint32_t	R_SCE_SHA1_HashUpdate (hash_ctrl_t *const p_ctrl, const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest) Computes the SHA1 message digest of the input message. More...
uint32_t	R_SCE_SHA256_Open (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg) SCE SHA256 HASH Initialization. More...
uint32_t	R_SCE_SHA256_Close (hash_ctrl_t *const p_ctrl) SCE SHA256 Close function. More...
uint32_t	R_SCE_SHA256_UpdateHash (const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest) Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest More...
uint32_t	R_SCE_SHA256_HashUpdate (hash_ctrl_t *const p_ctrl, const uint32_t *p_msg, uint32_t num_words, uint32_t *p_digest) Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest More...

Variables

const hash_api_t	g_md5_hash_on_sce
const hash_api_t	g_sha1_hash_on_sce
const hash_api_t	g_sha256_hash_on_sce

Detailed Description

Primitive cryptographic functions.

HASH functions

message hashing/digest functions

message hasing/digest functions

Function Documentation

◆ R_SCE_HASH_VersionGet()

uint32_t R_SCE_HASH_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

SCE HASH Get Version

Parameters

[in]	p_version	pointer to ssp_version_t structure where the version info will be stored

Return values

SF_CRYPTO_SUCCESS	Version information is returned successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ R_SCE_MD5_Close()

uint32_t R_SCE_MD5_Close (hash_ctrl_t *const *p_ctrl*)

SCE MD5 Close function.

Return values

SF_CRYPTO_SUCCESS	Module closed successfully.
SSP_ERR_ASSERTION	The input parameter is NULL.

◆ R_SCE_MD5_HashUpdate()

uint32_t R_SCE_MD5_HashUpdate (hash_ctrl_t *const *p_ctrl*, const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

MD5 HashUpdate Function.

Compute the MD5 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed MD5 message digest value.

Return values

SF_CRYPTO_SUCCESS	Function completed successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_ASSERTION	Input parameter p_digest is NULL.

Note

: The final message digest has to be byte swapped / the order of the bytes have to be
: reversed (the output of the intermediate updates do not have to be swapped).
: The message length has to be byte swapped (the endianness has to be reversed).

◆ R_SCE_MD5_Open()

uint32_t R_SCE_MD5_Open (hash_ctrl_t *const *p_ctrl*, hash_cfg_t const *const *p_cfg*)

SCE MD5 open function.

Return values

SF_CRYPTO_SUCCESS	Module opened successfully.
SSP_ERR_ASSERTION	At least one of the input parameters is NULL.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_MD5_UpdateHash()

uint32_t R_SCE_MD5_UpdateHash (const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

MD5 Update Hash function.

Compute the MD5 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed MD5 message digest value.

Return values

SF_CRYPTO_SUCCESS	Function completed successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_ASSERTION	At least one of the input parameters is NULL.

Note

: The final message digest has to be byte swapped / the order of the bytes have to be
: reversed (the output of the intermediate updates do not have to be swapped).
: The message length has to be byte swapped (the endianness has to be reversed).

◆ R_SCE_SHA1_Close()

uint32_t R_SCE_SHA1_Close (hash_ctrl_t *const *p_ctrl*)

SCE SHA1 Close function.

Return values

SF_CRYPTO_SUCCESS	Module is closed successfully.
SSP_ERR_ASSERTION	The input parameter is NULL.

◆ R_SCE_SHA1_HashUpdate()

uint32_t R_SCE_SHA1_HashUpdate (hash_ctrl_t *const *p_ctrl*, const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

Computes the SHA1 message digest of the input message.

Compute the SHA1 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed SHA1 message digest value.

Return values

SSP_ERR_ASSERTION	Input parameter p_digest is NULL.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

◆ R_SCE_SHA1_Open()

uint32_t R_SCE_SHA1_Open (hash_ctrl_t *const *p_ctrl*, hash_cfg_t const *const *p_cfg*)

SCE SHA1 open function using the SCE block.

SHA1 HASH Initialization

Return values

SF_CRYPTO_SUCCESS	open successful
SSP_ERR_ASSERTION	At least one of the input parameters is NULL.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_SHA1_UpdateHash()

uint32_t R_SCE_SHA1_UpdateHash (const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

Computes the SHA1 message digest of the input message.

Compute the SHA1 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed SHA1 message digest value.

Return values

SSP_ERR_ASSERTION	At least one of the input parameters is NULL.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

◆ R_SCE_SHA256_Close()

uint32_t R_SCE_SHA256_Close (hash_ctrl_t *const *p_ctrl*)

SCE SHA256 Close function.

Return values

SSP_ERR_ASSERTION	The input parameter is NULL.
SF_CRYPTO_SUCCESS	Module closed successfully.

◆ R_SCE_SHA256_HashUpdate()

uint32_t R_SCE_SHA256_HashUpdate (hash_ctrl_t *const *p_ctrl*, const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest

Compute the SHA256 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the contents of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value as given in buffer p_digest will be used and this buffer will be updated with the computed SHA256 message digest value.

Return values

SSP_ERR_ASSERTION	Input parameter p_digest is NULL.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

◆ R_SCE_SHA256_Open()

uint32_t R_SCE_SHA256_Open (hash_ctrl_t *const *p_ctrl*, hash_cfg_t const *const *p_cfg*)

SCE SHA256 HASH Initialization.

Return values

SSP_ERR_ASSERTION	At least one of the input parameters is NULL.
SF_CRYPTO_SUCCESS	successful completion.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_SHA256_UpdateHash()

uint32_t R_SCE_SHA256_UpdateHash (const uint32_t * *p_msg*, uint32_t *num_words*, uint32_t * *p_digest*)

Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest

Compute the SHA256 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the contents of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value as given in buffer p_digest will be used and this buffer will be updated with the computed SHA256 message digest value.

Return values

SSP_ERR_ASSERTION	At least one of the input parameters is NULL.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Variable Documentation

◆ g_md5_hash_on_sce

const hash_api_t g_md5_hash_on_sce

```
=
{
    .open       = R_SCE_MD5_Open,
    .updateHash = R_SCE_MD5_UpdateHash,
    .close      = R_SCE_MD5_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_MD5_HashUpdate
}
```

MD5 implementation of HASH API.

◆ g_sha1_hash_on_sce

```
const hash_api_t g_sha1_hash_on_sce
=
{
    .open       = R_SCE_SHA1_Open,
    .updateHash = R_SCE_SHA1_UpdateHash,
    .close      = R_SCE_SHA1_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_SHA1_HashUpdate
}
```

SHA1 implementation of HASH API.

◆ g_sha256_hash_on_sce

```
const hash_api_t g_sha256_hash_on_sce
=
{
    .open       = R_SCE_SHA256_Open,
    .updateHash = R_SCE_SHA256_UpdateHash,
    .close      = R_SCE_SHA256_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_SHA256_HashUpdate
}
```

SHA256 implementation of HASH API.

SCE_ECC

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

ssp_err_t	R_SCE_ECC_Open (ecc_ctrl_t *const p_ctrl, ecc_cfg_t const *const p_cfg) ECC Initialization. More...
ssp_err_t	R_SCE_ECC_Close (ecc_ctrl_t *const p_ctrl) ECC Close function. More...
ssp_err_t	R_SCE_ECC_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
ssp_err_t	R_SCE_ECC_192ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_ECC_192KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public) Generate public and private key pair for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_192PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_192PublicKeyVerify (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s)

	Signature verification of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_192ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_192KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_key_public) Generate public key and wrapped key for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_HRK_ECC_192PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_ECC_224ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_ECC_224KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public)

	Generate public and private key pair for elliptic curve cryptography on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_224PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_224PublicKeyVerify (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Signature verification of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_224ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_224KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_key_public) Generate public key and wrapped key for elliptic curve cryptography on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_HRK_ECC_224PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const

	p_key_index, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_ECC_256ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_ECC_256KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public) Generate public and private key pair for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_256PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_256PublicKeyVerify (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Signature verification of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...

ssp_err_t	R_SCE_HRK_ECC_256ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_256KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_key_public) Generate public key and wrapped key for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_HRK_ECC_256PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_ECC_384ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_k, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_ECC_384KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const p_key_public) Generate public and private key pair for elliptic curve cryptography on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...

ssp_err_t	R_SCE_ECC_384PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_private, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_ECC_384PublicKeyVerify (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_public, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Signature verification of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_384ScalarMultiplication (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_p, r_crypto_data_handle_t *const p_r) Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid. More...
ssp_err_t	R_SCE_HRK_ECC_384KeyCreate (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const p_key_public) Generate public key and wrapped key for elliptic curve cryptography on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...
ssp_err_t	R_SCE_HRK_ECC_384PrivateKeySign (ecc_ctrl_t *const p_ctrl, r_crypto_data_handle_t *const p_domain, r_crypto_data_handle_t *const p_generator_point, r_crypto_data_handle_t *const p_key_index, r_crypto_data_handle_t *const msg_digest, r_crypto_data_handle_t *const signature_r, r_crypto_data_handle_t *const signature_s) Generate signature of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the

	p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key. More...

Variables

const ecc_api_t	g_ecc192_on_sce
const ecc_api_t	g_ecc192_on_sce_hrk
const ecc_api_t	g_ecc224_on_sce
const ecc_api_t	g_ecc224_on_sce_hrk
const ecc_api_t	g_ecc256_on_sce
const ecc_api_t	g_ecc256_on_sce_hrk
const ecc_api_t	g_ecc384_on_sce
const ecc_api_t	g_ecc384_on_sce_hrk

Detailed Description

Primitive cryptographic functions.

ECC 192-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

ECC 224-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

ECC 256-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

ECC 384-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

Function Documentation

◆ R_SCE_ECC_192KeyCreate()

ssp_err_t R_SCE_ECC_192KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *p_key_public*)

Generate public and private key pair for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)
p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_key_private must have space to hold at least ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words) of output data.
p_key_public must have space to hold at least ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words) of output data.

◆ R_SCE_ECC_192PrivateKeySign()

ssp_err_t R_SCE_ECC_192PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)
p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_key_private must be of size ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words).
msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).
signature_r must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.
signature_s must have space to hold at least ECC_192_SIGNATURE_S_LENGTH_WORDS (6 words) of output data.

◆ R_SCE_ECC_192PublicKeyVerify()

ssp_err_t R_SCE_ECC_192PublicKeyVerify (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_public*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Signature verification of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)
p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_key_public must be of size ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words).
msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).
signature_r must be of size ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words).
signature_s must be of size ECC_192_SIGNATURE_S_LENGTH_WORDS (6 words).

◆ R_SCE_ECC_192ScalarMultiplication()

ssp_err_t R_SCE_ECC_192ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_k*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (18 words). (a||b||p - 6 words each)
p_k must be of size ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words).
p_p must be of size ECC_192_POINT_ON_CURVE_LENGTH_WORDS (12 words).
p_r must have space to hold at least ECC_192_POINT_ON_CURVE_LENGTH_WORDS (12 words) of output data.

◆ R_SCE_ECC_224KeyCreate()

ssp_err_t R_SCE_ECC_224KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *p_key_public*)

Generate public and private key pair for elliptic curve cryptography on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (28 words). (a||b||p||n - 7 words each)
p_generator_point must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_key_private must have space to hold at least ECC_224_PRIVATE_KEY_LENGTH_WORDS (7 words) of output data.
p_key_public must have space to hold at least ECC_224_PUBLIC_KEY_LENGTH_WORDS (14 words) of output data.

◆ R_SCE_ECC_224PrivateKeySign()

ssp_err_t R_SCE_ECC_224PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (28 words). (a||b||p||n - 7 words each)
p_generator_point must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_key_private must be of size ECC_224_PRIVATE_KEY_LENGTH_WORDS (7 words).
msg_digest must be of size ECC_224_MESSAGE_DIGEST_LENGTH_WORDS (7 words).
signature_r must have space to hold at least ECC_224_SIGNATURE_R_LENGTH_WORDS (7 words) of output data.
signature_s must have space to hold at least ECC_224_SIGNATURE_S_LENGTH_WORDS (7 words) of output data.

◆ R_SCE_ECC_224PublicKeyVerify()

ssp_err_t R_SCE_ECC_224PublicKeyVerify (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_public*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Signature verification of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (28 words). (a||b||p||n - 7 words each)
p_generator_point must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_key_public must be of size ECC_224_PUBLIC_KEY_LENGTH_WORDS (14 words).
msg_digest must be of size ECC_224_MESSAGE_DIGEST_LENGTH_WORDS (7 words).
signature_r must be of size ECC_224_SIGNATURE_R_LENGTH_WORDS (7 words).
signature_s must be of size ECC_224_SIGNATURE_S_LENGTH_WORDS (7 words).

◆ R_SCE_ECC_224ScalarMultiplication()

ssp_err_t R_SCE_ECC_224ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_k*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (21 words). (a||b||p - 7 words each)
p_k must be of size ECC_224_PRIVATE_KEY_LENGTH_WORDS (7 words).
p_p must be of size ECC_224_POINT_ON_CURVE_LENGTH_WORDS (14 words).
p_r must have space to hold at least ECC_224_POINT_ON_CURVE_LENGTH_WORDS (14 words) of output data.

◆ R_SCE_ECC_256KeyCreate()

ssp_err_t R_SCE_ECC_256KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *p_key_public*)

Generate public and private key pair for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)
p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).
p_key_private must have space to hold at least ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words) of output data.
p_key_public must have space to hold at least ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words) of output data.

◆ R_SCE_ECC_256PrivateKeySign()

ssp_err_t R_SCE_ECC_256PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)
p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).
p_key_private must be of size ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words).
msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).
signature_r must have space to hold at least ECC_256_SIGNATURE_R_LENGTH_WORDS (8 words) of output data.
signature_s must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.

◆ R_SCE_ECC_256PublicKeyVerify()

ssp_err_t R_SCE_ECC_256PublicKeyVerify (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_public*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Signature verification of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)
p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).
p_key_public must be of size ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words).
msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).
signature_r must be of size ECC_256_SIGNATURE_R_LENGTH_WORDS (8 words).
signature_s must be of size ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words).

◆ R_SCE_ECC_256ScalarMultiplication()

ssp_err_t R_SCE_ECC_256ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_k*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (24 words). (a||b||p - 8 words each)
p_k must be of size ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words).
p_p must be of size ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words).
p_r must have space to hold at least ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words) of output data.

◆ R_SCE_ECC_384KeyCreate()

ssp_err_t R_SCE_ECC_384KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *p_key_public*)

Generate public and private key pair for elliptic curve cryptography on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words). (a||b||p||n - 12 words each)
p_generator_point must be of size ECC_384_GENERATOR_POINT_LENGTH_WORDS (24 words).
p_key_private must have space to hold at least ECC_384_PRIVATE_KEY_LENGTH_WORDS (12 words) of output data.
p_key_public must have space to hold at least ECC_384_PUBLIC_KEY_LENGTH_WORDS (24 words) of output data.

◆ R_SCE_ECC_384PrivateKeySign()

ssp_err_t R_SCE_ECC_384PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_private*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words). (a||b||p||n - 12 words each)
p_generator_point must be of size ECC_384_GENERATOR_POINT_LENGTH_WORDS (24 words).
p_key_private must be of size ECC_384_PRIVATE_KEY_LENGTH_WORDS (12 words).
msg_digest must be of size ECC_384_MESSAGE_DIGEST_LENGTH_WORDS (12 words).
signature_r must have space to hold at least ECC_384_SIGNATURE_R_LENGTH_WORDS (12 words) of output data.
signature_s must have space to hold at least ECC_384_SIGNATURE_S_LENGTH_WORDS (12 words) of output data.

◆ R_SCE_ECC_384PublicKeyVerify()

ssp_err_t R_SCE_ECC_384PublicKeyVerify (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_public*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Signature verification of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words). (a||b||p||n - 12 words each)
p_generator_point must be of size ECC_384_GENERATOR_POINT_LENGTH_WORDS (24 words).
p_key_public must be of size ECC_384_PUBLIC_KEY_LENGTH_WORDS (24 words).
msg_digest must be of size ECC_384_MESSAGE_DIGEST_LENGTH_WORDS (12 words).
signature_r must be of size ECC_384_SIGNATURE_R_LENGTH_WORDS (12 words).
signature_s must be of size ECC_384_SIGNATURE_S_LENGTH_WORDS (12 words).

◆ R_SCE_ECC_384ScalarMultiplication()

ssp_err_t R_SCE_ECC_384ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_k*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (36 words). (a||b||p - 12 words each)
p_k must be of size ECC_384_PRIVATE_KEY_LENGTH_WORDS (12 words).
p_p must be of size ECC_384_POINT_ON_CURVE_LENGTH_WORDS (24 words).
p_r must have space to hold at least ECC_384_POINT_ON_CURVE_LENGTH_WORDS (24 words) of output data.

◆ R_SCE_ECC_Close()

ssp_err_t R_SCE_ECC_Close (ecc_ctrl_t *const *p_ctrl*)

ECC Close function.

Return values

SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

◆ R_SCE_ECC_Open()

ssp_err_t R_SCE_ECC_Open (ecc_ctrl_t *const *p_ctrl*, ecc_cfg_t const *const *p_cfg*)

ECC Initialization.

Name of module used by error logger macro

Return values

SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	Any of the input parameters are NULL.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_ECC_VersionGet()

ssp_err_t R_SCE_ECC_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ R_SCE_HRK_ECC_192KeyCreate()

ssp_err_t R_SCE_HRK_ECC_192KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_key_public*)

Generate public key and wrapped key for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)
p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_key_index must have space to hold at least ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words) of output data.
p_key_public must have space to hold at least ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words) of output data.

◆ R_SCE_HRK_ECC_192PrivateKeySign()

ssp_err_t R_SCE_HRK_ECC_192PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)
p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_key_index must be of size ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).
signature_r must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.
signature_s must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.

◆ R_SCE_HRK_ECC_192ScalarMultiplication()

ssp_err_t R_SCE_HRK_ECC_192ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

It is optional to concatenate the Order - 'n' to the 'p_domain' input buffer, i.e. both (a||b||p) and (a||b||p||n) are supported.
p_domain must be either of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words) - (a||b||p||n - 6 words each) OR of size ECC_192_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (18 words) - (a||b||p - 6 words each).
p_key_index must be of size ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
p_p must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).
p_r must have space to hold at least ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words) of output data.
p_key_index is generated using generateKey API.

◆ R_SCE_HRK_ECC_224KeyCreate()

ssp_err_t R_SCE_HRK_ECC_224KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_key_public*)

Generate public key and wrapped key for elliptic curve cryptography on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (28 words). (a||b||p||n - 7 words each)
p_generator_point must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_key_index must have space to hold at least ECC_224_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words) of output data.
p_key_public must have space to hold at least ECC_224_PUBLIC_KEY_LENGTH_WORDS (14 words) of output data.

◆ R_SCE_HRK_ECC_224PrivateKeySign()

ssp_err_t R_SCE_HRK_ECC_224PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 224 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_224_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (28 words). (a||b||p||n - 7 words each)
p_generator_point must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_key_index must be of size ECC_224_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
msg_digest must be of size ECC_224_MESSAGE_DIGEST_LENGTH_WORDS (7 words).
signature_r must have space to hold at least ECC_224_SIGNATURE_R_LENGTH_WORDS (7 words) of output data.
signature_s must have space to hold at least ECC_224_SIGNATURE_R_LENGTH_WORDS (7 words) of output data.

◆ R_SCE_HRK_ECC_224ScalarMultiplication()

ssp_err_t R_SCE_HRK_ECC_224ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

It is optional to concatenate the Order - 'n' to the 'p_domain' input buffer, i.e. both (a||b||p) and (a||b||p||n) are supported.
p_domain must be either of size ECC_224_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (28 words) - (a||b||p||n - 7 words each) OR of size ECC_224_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (21 words) - (a||b||p - 7 words each).
p_key_index must be of size ECC_224_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
p_p must be of size ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words).
p_r must have space to hold at least ECC_224_GENERATOR_POINT_LENGTH_WORDS (14 words) of output data.
p_key_index is generated using generateKey API.

◆ R_SCE_HRK_ECC_256KeyCreate()

ssp_err_t R_SCE_HRK_ECC_256KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_key_public*)

Generate public key and wrapped key for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)
p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).
p_key_index must have space to hold at least ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words) of output data.
p_key_public must have space to hold at least ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words) of output data.

◆ R_SCE_HRK_ECC_256PrivateKeySign()

ssp_err_t R_SCE_HRK_ECC_256PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)
p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).
p_key_index must be of size ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).
signature_r must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.
signature_s must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.

◆ R_SCE_HRK_ECC_256ScalarMultiplication()

ssp_err_t R_SCE_HRK_ECC_256ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

It is optional to concatenate the Order - 'n' to the 'p_domain' input buffer, i.e. both (a||b||p) and (a||b||p||n) are supported.
p_domain must be either of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words) - (a||b||p||n - 8 words each) OR of size ECC_256_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (24 words) - (a||b||p - 8 words each).
p_key_index must be of size ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).
p_p must be of size ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words).
p_r must have space to hold at least ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words) of output data.
p_key_index is generated using generateKey API.

◆ R_SCE_HRK_ECC_384KeyCreate()

ssp_err_t R_SCE_HRK_ECC_384KeyCreate (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_key_public*)

Generate public key and wrapped key for elliptic curve cryptography on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words). (a||b||p||n - 12 words each)
p_generator_point must be of size ECC_384_GENERATOR_POINT_LENGTH_WORDS (24 words).
p_key_index must have space to hold at least ECC_384_PRIVATE_KEY_HRK_LENGTH_WORDS (17 words) of output data.
p_key_public must have space to hold at least ECC_384_PUBLIC_KEY_LENGTH_WORDS (24 words) of output data.

◆ R_SCE_HRK_ECC_384PrivateKeySign()

ssp_err_t R_SCE_HRK_ECC_384PrivateKeySign (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_generator_point*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *msg_digest*, r_crypto_data_handle_t *const *signature_r*, r_crypto_data_handle_t *const *signature_s*)

Generate signature of ECDSA on 384 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

p_domain must be of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words). (a||b||p||n - 12 words each)
p_generator_point must be of size ECC_384_GENERATOR_POINT_LENGTH_WORDS (24 words).
p_key_index must be of size ECC_384_PRIVATE_KEY_HRK_LENGTH_WORDS (17 words).
msg_digest must be of size ECC_384_MESSAGE_DIGEST_LENGTH_WORDS (12 words).
signature_r must have space to hold at least ECC_384_SIGNATURE_S_LENGTH_WORDS (12 words) of output data.
signature_s must have space to hold at least ECC_384_SIGNATURE_S_LENGTH_WORDS (12 words) of output data.

◆ R_SCE_HRK_ECC_384ScalarMultiplication()

ssp_err_t R_SCE_HRK_ECC_384ScalarMultiplication (ecc_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t *const *p_domain*, r_crypto_data_handle_t *const *p_key_index*, r_crypto_data_handle_t *const *p_p*, r_crypto_data_handle_t *const *p_r*)

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Return values

SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

Note

In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will not be updated to the expected return buffer size and will be unchanged in case the API fails. The caller must check the return status before using the output data.

Precondition

SCE module must have been initialized by calling the functions R_SCE_Open()

Note

It is optional to concatenate the Order - 'n' to the 'p_domain' input buffer, i.e. both (a||b||p) and (a||b||p||n) are supported.
p_domain must be either of size ECC_384_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (48 words) - (a||b||p||n - 12 words each) OR of size ECC_384_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (36 words) - (a||b||p - 12 words each).
p_key_index must be of size ECC_384_PRIVATE_KEY_HRK_LENGTH_WORDS (17 words).
p_p must be of size ECC_384_POINT_ON_CURVE_LENGTH_WORDS (24 words).
p_r must have space to hold at least ECC_384_POINT_ON_CURVE_LENGTH_WORDS (24 words) of output data.
p_key_index is generated using generateKey API.

Variable Documentation

◆ g_ecc192_on_sce

```
const ecc_api_t g_ecc192_on_sce
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_ECC_192ScalarMultiplication,
    .keyCreate               = R_SCE_ECC_192KeyCreate,
    .sign                   = R_SCE_ECC_192PrivateKeySign,
    .verify                  = R_SCE_ECC_192PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc192_on_sce_hrk

```
const ecc_api_t g_ecc192_on_sce_hrk
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_HRK_ECC_192ScalarMultiplication,
    .keyCreate               = R_SCE_HRK_ECC_192KeyCreate,
    .sign                   = R_SCE_HRK_ECC_192PrivateKeySign,
    .verify                  = R_SCE_ECC_192PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc224_on_sce

const ecc_api_t g_ecc224_on_sce

```
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_ECC_224ScalarMultiplication,
    .keyCreate               = R_SCE_ECC_224KeyCreate,
    .sign                   = R_SCE_ECC_224PrivateKeySign,
    .verify                  = R_SCE_ECC_224PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc224_on_sce_hrk

const ecc_api_t g_ecc224_on_sce_hrk

```
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_HRK_ECC_224ScalarMultiplication,
    .keyCreate               = R_SCE_HRK_ECC_224KeyCreate,
    .sign                   = R_SCE_HRK_ECC_224PrivateKeySign,
    .verify                  = R_SCE_ECC_224PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc256_on_sce

```
const ecc_api_t g_ecc256_on_sce
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_ECC_256ScalarMultiplication,
    .keyCreate               = R_SCE_ECC_256KeyCreate,
    .sign                   = R_SCE_ECC_256PrivateKeySign,
    .verify                  = R_SCE_ECC_256PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc256_on_sce_hrk

```
const ecc_api_t g_ecc256_on_sce_hrk
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_HRK_ECC_256ScalarMultiplication,
    .keyCreate               = R_SCE_HRK_ECC_256KeyCreate,
    .sign                   = R_SCE_HRK_ECC_256PrivateKeySign,
    .verify                  = R_SCE_ECC_256PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc384_on_sce

```
const ecc_api_t g_ecc384_on_sce
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_ECC_384ScalarMultiplication,
    .keyCreate               = R_SCE_ECC_384KeyCreate,
    .sign                   = R_SCE_ECC_384PrivateKeySign,
    .verify                  = R_SCE_ECC_384PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

◆ g_ecc384_on_sce_hrk

```
const ecc_api_t g_ecc384_on_sce_hrk
=
{
    .open                   = R_SCE_ECC_Open,
    .close                   = R_SCE_ECC_Close,
    .scalarMultiplication    = R_SCE_HRK_ECC_384ScalarMultiplication,
    .keyCreate               = R_SCE_HRK_ECC_384KeyCreate,
    .sign                   = R_SCE_HRK_ECC_384PrivateKeySign,
    .verify                  = R_SCE_ECC_384PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

Exported global variablesSCE/ECC implementation of ECC API.

SCE_KEY_INSTALLATION

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Data Structures

struct	key_installation_instance_ctrl_t

Functions

ssp_err_t	R_SCE_KEY_INSTALLATION_Open (key_installation_ctrl_t *const p_ctrl, key_installation_cfg_t const *const p_cfg) Function to open the Key Installation module. More...
ssp_err_t	R_SCE_KEY_INSTALLATION_Close (key_installation_ctrl_t *const p_ctrl) Function to close the Key Installation module. More...
ssp_err_t	R_SCE_KEY_INSTALLATION_VersionGet (ssp_version_t *const p_version) Gets Key Installation Driver API and Code version based on compile time macros. More...
ssp_err_t	R_SCE_KEY_INSTALLATION_KeyInstall (key_installation_ctrl_t *const p_ctrl, r_crypto_data_handle_t const *const p_user_key_rsa_modulus, key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Key Installation function accepts the user's encrypted key, a shared index, session key, and IV then returns the wrapped key. More...
ssp_err_t	r_sce_key_installation_verify_shared_index (uint32_t const *const p_shared_index) Subroutine to verify that the shared index is using a valid enumeration for the Key Install API. More...
ssp_err_t	r_sce_key_installation_verify_session_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the user provided (input) session key size and buffer length. More...
ssp_err_t	r_sce_key_installation_aes (key_installation_key_t const *const

	p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Sub-routine to install user's AES encrypted key which is generated outside of the Synergy platform and processed using the DLM Service. More...
ssp_err_t	r_sce_key_installation_aes_192_bits (key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Sub-routine to install AES 192-bit encrypted user provided key which is generated outside of Synergy platform using the DLM Service, applies to ECB, GCM, CTR, CBC chaining modes. More...
ssp_err_t	r_sce_key_installation_aes_128_and_256_bits (key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Sub-routine to install AES 128/256-bit encrypted user provided key which is generated outside of Synergy platform with the DLM Service, applies to ECB, GCM, CTR, CBC and XTS chaining modes. More...
ssp_err_t	r_sce_key_installation_aes_xts (key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Subroutine to call AES Key installation hardware procedures for XTS chaining mode. More...
ssp_err_t	r_sce_key_installation_aes_verify_key_lengths (key_installation_key_t const *const p_user_key, key_installation_key_t const *const p_session_key, key_installation_key_t *const p_key_data) Subroutine to verify that the key_size and key lengths provided for the AES key, are valid to proceed with the Key Install API. More...
ssp_err_t	r_sce_key_installation_aes_verify_user_encrypted_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the user provided (input) encrypted AES key size and buffer length. More...
ssp_err_t	r_sce_key_installation_aes_verify_user_encrypted_key_length_helper (key_installation_key_size_t key_size, uint32_t data_length)

	Helper function to verify the user provided (input) encrypted AES key buffer length. More...
ssp_err_t	r_sce_key_installation_aes_verify_user_encrypted_xts_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the user provided (input) encrypted AES key buffer length. More...
ssp_err_t	r_sce_key_installation_aes_verify_out_wrapped_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the Output buffer installed key buffer length. More...
ssp_err_t	r_sce_key_installation_aes_verify_out_wrapped_xtskey_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the Output buffer installed key buffer length. More...
void	r_sce_key_installation_aes_fill_wrapped_key_out_length (key_installation_key_size_t key_size, uint32_t *data_length) Private helper function to fill the output wrapped key length. More...
ssp_err_t	r_sce_key_installation_ecc (key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Sub-routine to install ECC 192/256-bit encrypted user provided key which is generated outside of Synergy platform. More...
ssp_err_t	r_sce_key_installation_ecc_verify_key_lengths (key_installation_key_t const *const p_user_key, key_installation_key_t const *const p_session_key, key_installation_key_t *const p_key_data) Subroutine to verify that the provided key lengths are sufficient or not, before proceeding with key Installation API call. More...
ssp_err_t	r_sce_key_installation_ecc_verify_user_encrypted_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the user provided (input) encrypted ECC key size and buffer length. More...

ssp_err_t	r_sce_key_installation_ecc_verify_out_wrapped_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the Output buffer installed key buffer length. More...
void	r_sce_key_installation_ecc_fill_wrapped_key_out_length (key_installation_key_size_t key_size, uint32_t *data_length) Private helper function to fill the output wrapped key length. More...
ssp_err_t	r_sce_key_installation_rsa (r_crypto_data_handle_t const *const p_user_key_rsa_modulus, key_installation_key_t const *const p_user_key, key_installation_key_shared_index_t const shared_index, key_installation_key_t const *const p_session_key, uint32_t const *const p_iv, key_installation_key_t *const p_key_data) Sub-routine to install RSA 1024/2048-bit encrypted user provided key which is generated outside of Synergy platform. More...
ssp_err_t	r_sce_key_installation_rsa_verify_key_lengths (r_crypto_data_handle_t const *const p_user_key_rsa_modulus, key_installation_key_t const *const p_user_key, key_installation_key_t const *const p_session_key, key_installation_key_t *const p_key_data) Subroutine to verify that the provided key lengths are sufficient or not, before proceeding with key Installation API call. More...
ssp_err_t	r_sce_key_installation_rsa_verify_user_encrypted_key_length (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the user provided (input) encrypted RSA key buffer length. More...
void	r_sce_key_installation_rsa_fill_wrapped_key_out_length (key_installation_key_size_t key_size, uint32_t *data_length) Private helper function to fill the output wrapped key length. More...
ssp_err_t	r_sce_key_installation_rsa_verify_out_wrapped_key_length_including_modulus (key_installation_key_size_t key_size, uint32_t data_length) Subroutine to verify the length of (output buffer) installed key buffer. This length includes exponent, modulus and extra space for key wrapping. More...
ssp_err_t	r_sce_key_installation_rsa_verify_modulus (key_installation_key_size_t key_size, uint32_t data_length)

Subroutine to verify the modulus of RSA key. More...

Variables

const key_installation_api_t	g_key_installation_on_sce

Detailed Description

Primitive cryptographic functions.

key installation functions

Function Documentation

◆ r_sce_key_installation_aes()

ssp_err_t r_sce_key_installation_aes (key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Sub-routine to install user's AES encrypted key which is generated outside of the Synergy platform and processed using the DLM Service.

Parameters

[in]	p_user_key	Pointer to a user provided AES encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed AES key.

Return values

SSP_SUCCESS	AES Key Installation is successful.
SSP_ERR_INVALID_ARGUMENT	Input key size is invalid.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input key buffer lengths are invalid.

◆ r_sce_key_installation_aes_128_and_256_bits()

ssp_err_t r_sce_key_installation_aes_128_and_256_bits (key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Sub-routine to install AES 128/256-bit encrypted user provided key which is generated outside of Synergy platform with the DLM Service, applies to ECB, GCM, CTR, CBC and XTS chaining modes.

Parameters

[in]	p_user_key	Pointer to a user provided AES encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed AES key.

Return values

SSP_SUCCESS	AES Key Installation is successful.
SSP_ERR_INVALID_ARGUMENT	Input key size is invalid.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input key buffer lengths are invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

◆ r_sce_key_installation_aes_192_bits()

ssp_err_t r_sce_key_installation_aes_192_bits (key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Sub-routine to install AES 192-bit encrypted user provided key which is generated outside of Synergy platform using the DLM Service, applies to ECB, GCM, CTR, CBC chaining modes.

Parameters

[in]	p_user_key	Pointer to a user provided AES encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed AES key.

Return values

SSP_SUCCESS	AES Key Installation is successful.
SSP_ERR_INVALID_ARGUMENT	Input key size is invalid.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input key buffer lengths are invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

◆ r_sce_key_installation_aes_fill_wrapped_key_out_length()

void r_sce_key_installation_aes_fill_wrapped_key_out_length (key_installation_key_size_t *key_size*, uint32_t * *data_length*)

Private helper function to fill the output wrapped key length.

Parameters

[in]	key_size	Indicates users encrypted AES key sizes - 128/192/256-bit.
[in,out]	data_length	Pointer to the length of the wrapped key data in the buffer in WORDS, filled by this routine.

Note

The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_aes_verify_key_lengths()

ssp_err_t r_sce_key_installation_aes_verify_key_lengths (key_installation_key_t const *const *p_user_key*, key_installation_key_t const *const *p_session_key*, key_installation_key_t *const *p_key_data*)

Subroutine to verify that the key_size and key lengths provided for the AES key, are valid to proceed with the Key Install API.

Parameters

[in]	p_user_key	Indicates AES key sizes - 128/192/256-bits, supported chaining modes CBC, ECB, GCM, CTR. 128/256-bits, supported chaining modes - XTS.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service.
[in]	p_key_data	Pointer to wrapped installed AES key.

Return values

SSP_SUCCESS	Key lengths (user provided encrypted input key and output wrapped buffer which holds installed key) are sufficient, and can proceed with key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided key_size is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	Either Input key buffer lengths or Output key buffer length is not valid for AES Key installation API operation.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed key buffer lengths are sufficient for Key Installation API call.

◆ r_sce_key_installation_aes_verify_out_wrapped_key_length()

ssp_err_t r_sce_key_installation_aes_verify_out_wrapped_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the Output buffer installed key buffer length.

Parameters

[in]	key_size	Indicates user's encrypted AES key sizes - 128/192/256-bits.
[in]	data_length	Length of output wrapped key data in the buffer in WORDS.

Return values

SSP_SUCCESS	Output wrapped Key buffer length is sufficient and can proceed with Key installation API call.
SSP_ERR_CRYPTO_INVALID_SIZE	Output wrapped key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call. The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_aes_verify_out_wrapped_xtskey_length()

ssp_err_t r_sce_key_installation_aes_verify_out_wrapped_xtskey_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the Output buffer installed key buffer length.

Parameters

[in]	key_size	Indicates user's encrypted AES key sizes - 128/256-bit XTS key.
[in]	data_length	Length of wrapped key data in the buffer in WORDS.

Return values

SSP_SUCCESS	Output wrapped Key buffer length is sufficient and can proceed with Key installation API call.
SSP_ERR_CRYPTO_INVALID_SIZE	Output wrapped key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call. The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_aes_verify_user_encrypted_key_length()

ssp_err_t r_sce_key_installation_aes_verify_user_encrypted_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the user provided (input) encrypted AES key size and buffer length.

Parameters

[in]	key_size	Indicates AES key sizes - 128/192/256-bits.
[in]	data_length	Length of AES encrypted key data buffer in WORDS..

Return values

SSP_SUCCESS	User provided encrypted input Key length is sufficient and can proceed with Key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided Input encrypted key size argument is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call.

◆ r_sce_key_installation_aes_verify_user_encrypted_key_length_helper()

ssp_err_t r_sce_key_installation_aes_verify_user_encrypted_key_length_helper (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Helper function to verify the user provided (input) encrypted AES key buffer length.

Parameters

[in]	key_size	Indicates AES key sizes - 128/192/256-bits.
[in]	data_length	Length of AES encrypted key data buffer in WORDS.

Return values

SSP_SUCCESS	User provided encrypted input Key buffer length is sufficient and can proceed with Key installation API call.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is invalid.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call. The user key_size should be validated before this private function is called. The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_aes_verify_user_encrypted_xts_key_length()

ssp_err_t r_sce_key_installation_aes_verify_user_encrypted_xts_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the user provided (input) encrypted AES key buffer length.

Parameters

[in]	key_size	Indicates AES key sizes - 128/256-bit XTS.
[in]	data_length	Length of AES encrypted key data buffer in WORDS.

Return values

SSP_SUCCESS	User provided encrypted input Key length is sufficient and can proceed with Key installation API call.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is invalid.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call. The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_aes_xts()

ssp_err_t r_sce_key_installation_aes_xts (key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Subroutine to call AES Key installation hardware procedures for XTS chaining mode.

Parameters

[in]	p_user_key	Pointer to a user provided AES encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed AES key.

Return values

SSP_SUCCESS	AES Key Installation is successful.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input key buffer lengths are invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

◆ R_SCE_KEY_INSTALLATION_Close()

ssp_err_t R_SCE_KEY_INSTALLATION_Close (key_installation_ctrl_t *const *p_ctrl*)

Function to close the Key Installation module.

Return values

SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	NULL input parameter(s).

Note

This API must be called once Key Installation is done.

◆ r_sce_key_installation_ecc()

ssp_err_t r_sce_key_installation_ecc (key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Sub-routine to install ECC 192/256-bit encrypted user provided key which is generated outside of Synergy platform.

Parameters

[in]	p_user_key	Pointer to a user provided ECC encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed ECC key.

Return values

SF_CRYPTO_SUCCESS	ECC Key Installation is successful.
SSP_ERR_INVALID_ARGUMENT	Input key size is invalid.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input key buffer lengths are invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

◆ r_sce_key_installation_ecc_fill_wrapped_key_out_length()

void r_sce_key_installation_ecc_fill_wrapped_key_out_length (key_installation_key_size_t *key_size*, uint32_t * *data_length*)

Private helper function to fill the output wrapped key length.

Parameters

[in]	key_size	Indicates users encrypted ECC key sizes - 192/256-bit.
[in,out]	data_length	Pointer to the length of the wrapped key data in the buffer in WORDS, filled by this routine.

Note

The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_ecc_verify_key_lengths()

ssp_err_t r_sce_key_installation_ecc_verify_key_lengths (key_installation_key_t const *const *p_user_key*, key_installation_key_t const *const *p_session_key*, key_installation_key_t *const *p_key_data*)

Subroutine to verify that the provided key lengths are sufficient or not, before proceeding with key Installation API call.

Parameters

[in]	p_user_key	Indicates ECC key sizes - 192/224/256/384-bits.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service.
[in]	p_key_data	Pointer to wrapped installed ECC key.

Return values

SF_CRYPTO_SUCCESS	Key lengths (user provided encrypted input key, Renesas provided install key, and output wrapped buffer which holds installed key) are sufficient, and can proceed with key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided Input encrypted key size argument is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	Either Input key buffer lengths or Output key buffer length is not valid for ECC Key installation API operation.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed key buffer lengths are sufficient for Key Installation API call.

◆ r_sce_key_installation_ecc_verify_out_wrapped_key_length()

ssp_err_t r_sce_key_installation_ecc_verify_out_wrapped_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the Output buffer installed key buffer length.

Parameters

[in]	key_size	Indicates user's encrypted ECC key sizes - 192/256-bit.
[in]	data_length	The length of the output wrapped key data in the buffer in WORDS.

Return values

SF_CRYPTO_SUCCESS	Output wrapped Key buffer length is sufficient and can proceed with Key installation API call.
SSP_ERR_CRYPTO_INVALID_SIZE	Output wrapped key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call. The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_ecc_verify_user_encrypted_key_length()

ssp_err_t r_sce_key_installation_ecc_verify_user_encrypted_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the user provided (input) encrypted ECC key size and buffer length.

Parameters

[in]	key_size	Indicates ECC key sizes.
[in]	data_length	The length of the encrypted ECC key data in the buffer in WORDS.

Return values

SF_CRYPTO_SUCCESS	User provided encrypted input Key size and buffer length is valid and can proceed with Key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided Input encrypted key size argument is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is invalid.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call.

◆ R_SCE_KEY_INSTALLATION_KeyInstall()

ssp_err_t R_SCE_KEY_INSTALLATION_KeyInstall (key_installation_ctrl_t *const *p_ctrl*, r_crypto_data_handle_t const *const *p_user_key_rsa_modulus*, key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Key Installation function accepts the user's encrypted key, a shared index, session key, and IV then returns the wrapped key.

Return values

SSP_SUCCESS	Successful Key install.
SSP_ERR_ASSERTION	The parameters p_ctrl or p_user_key or p_session_key or p_iv or p_key_data is NULL.
SSP_ERR_INVALID_ARGUMENT	If one or more conditions below are true - 1. Input key_size of user provided encrypted key 2. In case of RSA key installation, p_user_key_rsa_modulus is set to NULL. 3. In case of RSA key installation, p_data field of p_user_key_rsa_modulus is set to NULL.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	RSA/ ECC execution is not yet available.
SSP_ERR_CRYPTO_INVALID_SIZE	Incoming buffer length (user provided encrypted key or session key or output key data) is invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

Note

It is the responsibility of the caller to store the key.
For key installations other than RSA, p_user_key_rsa_modulus should be set to NULL.

◆ R_SCE_KEY_INSTALLATION_Open()

ssp_err_t R_SCE_KEY_INSTALLATION_Open (key_installation_ctrl_t *const *p_ctrl*, key_installation_cfg_t const *const *p_cfg*)

Function to open the Key Installation module.

Parameters

[in,out]	p_ctrl	Pointer to Key Installation control block.
[in]	p_cfg	Pointer to Key Installation configuration structure.

Return values

SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	NULL input parameter(s).

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ r_sce_key_installation_rsa()

ssp_err_t r_sce_key_installation_rsa (r_crypto_data_handle_t const *const *p_user_key_rsa_modulus*, key_installation_key_t const *const *p_user_key*, key_installation_key_shared_index_t const *shared_index*, key_installation_key_t const *const *p_session_key*, uint32_t const *const *p_iv*, key_installation_key_t *const *p_key_data*)

Sub-routine to install RSA 1024/2048-bit encrypted user provided key which is generated outside of Synergy platform.

Parameters

[in]	p_user_key_rsa_modulus	RSA key modulus.
[in]	p_user_key	Pointer to a user provided RSA encrypted key.
[in]	shared_index	Shared Key Index that is returned by the DLM Service, accompanied by the Session Key that follows.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service, accompanied by the previous Shared Key Index parameter.
[in]	p_iv	Pointer to the IV used to encrypt the User Key.
[in,out]	p_key_data	Pointer to wrapped installed RSA key.

Return values

SSP_SUCCESS	RSA Key Installation is successful.
SSP_ERR_INVALID_ARGUMENT	Input key size is invalid.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	RSA modulus or input key buffer lengths are invalid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_SCE_FAIL	Key Installation failed.

◆ r_sce_key_installation_rsa_fill_wrapped_key_out_length()

void r_sce_key_installation_rsa_fill_wrapped_key_out_length (key_installation_key_size_t *key_size*, uint32_t * *data_length*)

Private helper function to fill the output wrapped key length.

Parameters

[in]	key_size	Indicates users encrypted RSA key sizes - 1024/2048-bit.
[in,out]	data_length	Pointer to the length of the wrapped key data in the buffer in WORDS, filled by this routine.

Note

The user key_size should be validated before this private function is called.

◆ r_sce_key_installation_rsa_verify_key_lengths()

ssp_err_t r_sce_key_installation_rsa_verify_key_lengths (r_crypto_data_handle_t const *const *p_user_key_rsa_modulus*, key_installation_key_t const *const *p_user_key*, key_installation_key_t const *const *p_session_key*, key_installation_key_t *const *p_key_data*)

Subroutine to verify that the provided key lengths are sufficient or not, before proceeding with key Installation API call.

Parameters

[in]	p_user_key_rsa_modulus	RSA key modulus.
[in]	p_user_key	Indicates RSA key sizes - 1024/2048-bits.
[in]	p_session_key	Pointer to the Session Key returned by the DLM Service.
[in]	p_key_data	Pointer to wrapped installed RSA key.

Return values

SSP_SUCCESS	Key lengths (user provided encrypted input key, Renesas provided install key, and output wrapped buffer which holds installed key) are sufficient, and can proceed with key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided key_size is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	Either Input key buffer lengths or Output key buffer length is not valid for RSA Key installation API operation.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed key buffer lengths are sufficient for Key Installation API call.

◆ r_sce_key_installation_rsa_verify_modulus()

ssp_err_t r_sce_key_installation_rsa_verify_modulus (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the modulus of RSA key.

Parameters

[in]	key_size	Size of user key to be installed.
[in]	data_length	Length of user's input RSA modulus, 1024/2048-bit

Return values

SSP_ERR_CRYPTO_INVALID_SIZE	Invalid modulus length.
SSP_ERR_INVALID_ARGUMENT	This is the return code for one or more cases below - 1. Invalid Key Size is provided.
SSP_SUCCESS	Valid modulus length.

◆ r_sce_key_installation_rsa_verify_out_wrapped_key_length_including_modulus()

ssp_err_t r_sce_key_installation_rsa_verify_out_wrapped_key_length_including_modulus (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the length of (output buffer) installed key buffer. This length includes exponent, modulus and extra space for key wrapping.

Parameters

[in]	key_size	Size of user key to be installed.
[in]	data_length	Length of user's input RSA modulus, 1024/2048-bit.

Return values

SSP_ERR_CRYPTO_INVALID_SIZE	Output wrapped key buffer length is not sufficient.
SSP_ERR_INVALID_ARGUMENT	Invalid Key Size is provided.
SSP_SUCCESS	Output wrapped key buffer length has sufficient space to store the returned key.

◆ r_sce_key_installation_rsa_verify_user_encrypted_key_length()

ssp_err_t r_sce_key_installation_rsa_verify_user_encrypted_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the user provided (input) encrypted RSA key buffer length.

Parameters

[in]	key_size	Indicates RSA key sizes - 1024/2048-bits.
[in]	data_length	The length of the encrypted RSA key data in the buffer in WORDS.

Return values

SSP_SUCCESS	User provided encrypted input Key length is sufficient and can proceed with Key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided Input encrypted key size argument is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call.

◆ r_sce_key_installation_verify_session_key_length()

ssp_err_t r_sce_key_installation_verify_session_key_length (key_installation_key_size_t *key_size*, uint32_t *data_length*)

Subroutine to verify the user provided (input) session key size and buffer length.

Parameters

[in]	key_size	Indicates the expected session key size for all operations.
[in]	data_length	Length of session key data buffer in WORDS.

Return values

SSP_SUCCESS	User provided encrypted input Key length is sufficient and can proceed with Key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided Input encrypted key size argument is invalid.
SSP_ERR_CRYPTO_INVALID_SIZE	User provided Input encrypted key buffer length is not sufficient.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed user key buffer length is sufficient for Key Installation API call.

◆ r_sce_key_installation_verify_shared_index()

ssp_err_t r_sce_key_installation_verify_shared_index (uint32_t const *const *p_shared_index*)

Subroutine to verify that the shared index is using a valid enumeration for the Key Install API.

Parameters

[in]	p_shared_index	Shared Key Index that is returned by the DLM Service.

Return values

SSP_SUCCESS	Shared Index is sufficient, can proceed with key installation API call.
SSP_ERR_INVALID_ARGUMENT	User provided an invalid shared_index value.

Note

This function is not a user API but an internal function for "keyInstallation" API to verify whether passed shared index is sufficient for Key Installation API call.

◆ R_SCE_KEY_INSTALLATION_VersionGet()

ssp_err_t R_SCE_KEY_INSTALLATION_VersionGet (ssp_version_t *const *p_version*)

Gets Key Installation Driver API and Code version based on compile time macros.

Return values

SSP_SUCCESS	Successful in getting the module version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_key_installation_on_sce

const key_installation_api_t g_key_installation_on_sce

```
=
{
    .open            = R_SCE_KEY_INSTALLATION_Open,
    .close          = R_SCE_KEY_INSTALLATION_Close,
    .versionGet      = R_SCE_KEY_INSTALLATION_VersionGet,
    .keyInstall      = R_SCE_KEY_INSTALLATION_KeyInstall
}
```

SCE/Key_Installation API implementation.

key_installation_instance_ctrl_t Struct Reference

Renesas Synergy Software Package Reference » HAL Layer » SCE Module » SCE_KEY_INSTALLATION

#include <r_sce_key_installation.h>

Data Fields

crypto_ctrl_t *	p_crypto_ctrl pointer to crypto engine control structure
crypto_api_t const *	p_crypto_api pointer to crypto engine API

Detailed Description

Key Installation Interface control structure

The documentation for this struct was generated from the following file:

- r_sce_key_installation.h

SCE_RSA

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Macros

#define	RSA_KEYLENGTH (1024U)
#define	UINT32_BITS (32)
#define	UINT32_BYTES (4)
#define	RSA_MODULUS_SIZE ((RSA_KEYLENGTH)/(UINT32_BITS))
#define	RSA_PRIVATE_EXPONENT_SIZE (RSA_MODULUS_SIZE)
#define	RSA_PUBLIC_EXPONENT_SIZE (1)
#define	RSA_PUBLIC_KEYSIZE ((RSA_PUBLIC_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
#define	RSA_PRIVATE_KEYSIZE ((RSA_PRIVATE_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
#define	RSA_PRIVATE_CRT_KEYSIZE ((5*(RSA_MODULUS_SIZE))/2)
#define	RSA_KEYLENGTH (2048U)
#define	UINT32_BITS (32)
#define	UINT32_BYTES (4)
#define	RSA_MODULUS_SIZE ((RSA_KEYLENGTH)/(UINT32_BITS))

#define	RSA_PRIVATE_EXPONENT_SIZE (RSA_MODULUS_SIZE)
#define	RSA_PUBLIC_EXPONENT_SIZE (1)
#define	RSA_PUBLIC_KEYSIZE ((RSA_PUBLIC_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
#define	RSA_PRIVATE_KEYSIZE ((RSA_PRIVATE_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
#define	RSA_PRIVATE_CRT_KEYSIZE ((5*(RSA_MODULUS_SIZE))/2)

Functions

uint32_t	R_SCE_RSA_Open (rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)
uint32_t	R_SCE_RSA_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_RSA_Close (rsa_ctrl_t *const p_ctrl)
uint32_t	R_SCE_HRK_RSA_1024PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt using 1024-bit RSA public key. More...
uint32_t	R_SCE_HRK_RSA_1024PublicKeyVerify (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash) Signature Verification using 1024-bit RSA public key. More...
uint32_t	R_SCE_HRK_RSA_1024PrivateKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 1024-bit RSA private key in wrapped format. More...
uint32_t	R_SCE_HRK_RSA_1024PrivateKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 1024-bit RSA private key in wrapped format. More...
uint32_t	R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt (rsa_ctrl_t *const p_ctrl,

	const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 1024-bit RSA wrapped private key in CRT format. More...
uint32_t	R_SCE_HRK_RSA_1024PrivateCrtKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 1024-bit RSA wrapped private key represented in CRT format. More...
uint32_t	R_SCE_HRK_RSA_1024KeyCreate (rsa_ctrl_t *const p_ctrl, rsa_key_t *p_private_key, rsa_key_t *p_public_key) Creation of 1024-bit RSA Key pair in with private key in wrapped format. More...
uint32_t	R_SCE_HRK_RSA_2048PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encrypt using 2048-bit RSA public key. More...
uint32_t	R_SCE_HRK_RSA_2048PublicKeyVerify (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash) Signature Verification using 2048-bit RSA public key. More...
uint32_t	R_SCE_HRK_RSA_2048PrivateKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 2048-bit RSA private key in wrapped format. More...
uint32_t	R_SCE_HRK_RSA_2048PrivateKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 2048-bit RSA private key in wrapped format. More...
uint32_t	R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 2048-bit RSA wrapped private key in CRT format.

	More...
uint32_t	R_SCE_HRK_RSA_2048PrivateCrtKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 2048-bit RSA wrapped private key represented in CRT format. More...
uint32_t	R_SCE_HRK_RSA_2048KeyCreate (rsa_ctrl_t *const p_ctrl, rsa_key_t *p_private_key, rsa_key_t *p_public_key) Creation of 2048-bit RSA Key pair in with private key in wrapped format. More...
uint32_t	R_SCE_RSA_1024PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Encryption using 1024-bit RSA public key. More...
uint32_t	R_SCE_RSA_1024PublicKeyVerify (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash) Signature Verification using 1024-bit RSA public key. More...
uint32_t	R_SCE_RSA_1024PrivateKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 1024-bit RSA private key (standard format) More...
uint32_t	R_SCE_RSA_1024PrivateKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 1024-bit RSA private key. More...
uint32_t	R_SCE_RSA_1024PrivateCrtKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 1024-bit RSA private key (CRT format) More...
uint32_t	R_SCE_RSA_1024PrivateCrtKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt,

	uint32_t *p_source, uint32_t *p_dest) Signature generation using 1024-bit RSA private key represented in CRT format. More...
uint32_t	R_SCE_RSA_1024KeyCreate (rsa_ctrl_t *const p_ctrl, rsa_key_t *p_private_key, rsa_key_t *p_public_key) Creation of 1024-bit RSA Key pair in plain text format. More...
uint32_t	R_SCE_RSA_2048PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Encrypt using 2048-bit RSA public key. More...
uint32_t	R_SCE_RSA_2048PublicKeyVerify (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_signature, uint32_t *p_paddedHash) Signature Verification using 2048-bit RSA public key. More...
uint32_t	R_SCE_RSA_2048PrivateKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 2048-bit RSA private key (standard format) More...
uint32_t	R_SCE_RSA_2048PrivateKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 2048-bit RSA private key. More...
uint32_t	R_SCE_RSA_2048PrivateCrtKeyDecrypt (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Decrypt using 2048-bit RSA private key (CRT format) More...
uint32_t	R_SCE_RSA_2048PrivateCrtKeySign (rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest) Signature generation using 2048-bit RSA private key represented in CRT format. More...
uint32_t	R_SCE_RSA_2048KeyCreate (rsa_ctrl_t *const p_ctrl, rsa_key_t

	*p_private_key, rsa_key_t *p_public_key) Creation of 2048-bit RSA Key pair in plain text format. More...

Variables

const rsa_api_t	g_rsa1024_on_sce_hrk
const rsa_api_t	g_rsa2048_on_sce_hrk
const rsa_api_t	g_rsa1024_on_sce

Detailed Description

Primitive cryptographic functions.

RSA 1024-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for wrapped keys.

RSA 2048-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for wrapped keys.

RSA 1024-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for plain-text/ raw keys.

RSA 2048-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for plain-text/ raw keys.

RSA common functions

Macro Definition Documentation

◆ RSA_KEYLENGTH [1/2]

#define RSA_KEYLENGTH (1024U)
RSA Keylength in bits

◆ RSA_KEYLENGTH [2/2]

#define RSA_KEYLENGTH (2048U)
RSA Keylength in bits

◆ RSA_MODULUS_SIZE [1/2]

#define RSA_MODULUS_SIZE ((RSA_KEYLENGTH)/(UINT32_BITS))
rsa modulus data size in words

◆ RSA_MODULUS_SIZE [2/2]

#define RSA_MODULUS_SIZE ((RSA_KEYLENGTH)/(UINT32_BITS))
rsa modulus data size in words

◆ RSA_PRIVATE_CRT_KEYSIZE [1/2]

#define RSA_PRIVATE_CRT_KEYSIZE ((5*(RSA_MODULUS_SIZE))/2)
RSA private key in CRT format size in words

◆ RSA_PRIVATE_CRT_KEYSIZE [2/2]

#define RSA_PRIVATE_CRT_KEYSIZE ((5*(RSA_MODULUS_SIZE))/2)
RSA private key in CRT format size in words

◆ RSA_PRIVATE_EXPONENT_SIZE [1/2]

#define RSA_PRIVATE_EXPONENT_SIZE (RSA_MODULUS_SIZE)
rsa private exponent data size in words

◆ RSA_PRIVATE_EXPONENT_SIZE [2/2]

#define RSA_PRIVATE_EXPONENT_SIZE (RSA_MODULUS_SIZE)
rsa private exponent data size in words

◆ RSA_PRIVATE_KEYSIZE [1/2]

#define RSA_PRIVATE_KEYSIZE ((RSA_PRIVATE_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
rsa private key size in words

◆ RSA_PRIVATE_KEYSIZE [2/2]

#define RSA_PRIVATE_KEYSIZE ((RSA_PRIVATE_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
rsa private key size in words

◆ RSA_PUBLIC_EXPONENT_SIZE [1/2]

```
#define RSA_PUBLIC_EXPONENT_SIZE  (1)
```

rsa public exponent data size in words

◆ RSA_PUBLIC_EXPONENT_SIZE [2/2]

```
#define RSA_PUBLIC_EXPONENT_SIZE  (1)
```

rsa public exponent data size in words

◆ RSA_PUBLIC_KEYSIZE [1/2]

```
#define RSA_PUBLIC_KEYSIZE  ((RSA_PUBLIC_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
```

rsa public key size in words

◆ RSA_PUBLIC_KEYSIZE [2/2]

```
#define RSA_PUBLIC_KEYSIZE  ((RSA_PUBLIC_EXPONENT_SIZE)+(RSA_MODULUS_SIZE))
```

rsa public key size in words

◆ UINT32_BITS [1/2]

```
#define UINT32_BITS  (32)
```

uint32_t word size in bits

◆ UINT32_BITS [2/2]

```
#define UINT32_BITS  (32)
```

uint32_t word size in bits

◆ UINT32_BYTES [1/2]

```
#define UINT32_BYTES  (4)
```

uint32_t word size in bytes

◆ UINT32_BYTES [2/2]

#define UINT32_BYTES (4)
uint32_t word size in bytes

Function Documentation

◆ R_SCE_HRK_RSA_1024KeyCreate()

uint32_t R_SCE_HRK_RSA_1024KeyCreate (rsa_ctrl_t *const *p_ctrl*, rsa_key_t * *p_private_key*, rsa_key_t * *p_public_key*)

Creation of 1024-bit RSA Key pair in with private key in wrapped format.

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY specifies a wrapped standard format key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_CRT_KEY specifies a wrapped CRT format key.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not be empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

Note

The buffers to hold the keys must be adequate for the key formats requested.

◆ R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt()

uint32_t R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 1024-bit RSA wrapped private key in CRT format.

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA wrapped private key in CRT format, from buffer key . The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format/size.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
'p_domain' is unused and can be passed as NULL.
The p_key pointer to 85-word buffer with Wrapped 1024-bit RSA key CRT parameters.
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_HRK_RSA_1024PrivateCrtKeySign()

uint32_t R_SCE_HRK_RSA_1024PrivateCrtKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 1024-bit RSA wrapped private key represented in CRT format.

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA wrapped private key in CRT format, from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
'p_domain' is unused and can be passed as NULL.
The p_key pointer to 85-word buffer with 1024-bit RSA key CRT parameters
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_HRK_RSA_1024PrivateKeyDecrypt()

uint32_t R_SCE_HRK_RSA_1024PrivateKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 1024-bit RSA private key in wrapped format.

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Decrypt operation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/32 = 32 words.

◆ R_SCE_HRK_RSA_1024PrivateKeySign()

uint32_t R_SCE_HRK_RSA_1024PrivateKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 1024-bit RSA private key in wrapped format.

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Signature generation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/32 = 32 words.

◆ R_SCE_HRK_RSA_1024PublicKeyEncrypt()

uint32_t R_SCE_HRK_RSA_1024PublicKeyEncrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt using 1024-bit RSA public key.

Encrypt num_words words of input data from buffer p_source using the 1024-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 32 words of data
num_words must be equal to RSA MODULUS size in bytes (RSA Public Key_size / bits per byte). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/8 = 128 bytes or 32 words.

◆ R_SCE_HRK_RSA_1024PublicKeyVerify()

uint32_t R_SCE_HRK_RSA_1024PublicKeyVerify (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_signature*, uint32_t * *p_padded_hash*)

Signature Verification using 1024-bit RSA public key.

Verify RSA signature data from buffer p_signature of length num_words using 1024-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature or/and Input parameter p_padded_hash is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 32 words.
Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 1024-bit key, num_words should be equal to 32.
num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/32 = 32 words.

◆ R_SCE_HRK_RSA_2048KeyCreate()

uint32_t R_SCE_HRK_RSA_2048KeyCreate (rsa_ctrl_t *const *p_ctrl*, rsa_key_t * *p_private_key*, rsa_key_t * *p_public_key*)

Creation of 2048-bit RSA Key pair in with private key in wrapped format.

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY specifies a wrapped standard format key.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY specifies the CRT parameters to be created.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not be empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

Note

The buffers to hold the keys must be adequate for the key formats requested.

◆ R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt()

uint32_t R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 2048-bit RSA wrapped private key in CRT format.

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key in Wrapped CRT format, from buffer key . The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format/size.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
'p_domain' is unused and can be passed as NULL.
The p_key pointer to 165-word buffer with Wrapped 1024-bit RSA key CRT parameters.
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 2048/32 = 64 words.

◆ R_SCE_HRK_RSA_2048PrivateCrtKeySign()

uint32_t R_SCE_HRK_RSA_2048PrivateCrtKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 2048-bit RSA wrapped private key represented in CRT format.

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA wrapped private key in CRT format, from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
'p_domain' is unused and can be passed as NULL.
The p_key pointer to 165-word buffer with 2048-bit RSA key CRT parameters
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 2048/32 = 64 words.

◆ R_SCE_HRK_RSA_2048PrivateKeyDecrypt()

uint32_t R_SCE_HRK_RSA_2048PrivateKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 2048-bit RSA private key in wrapped format.

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Decrypt operation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

◆ R_SCE_HRK_RSA_2048PrivateKeySign()

uint32_t R_SCE_HRK_RSA_2048PrivateKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 2048-bit RSA private key in wrapped format.

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Signature generation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

◆ R_SCE_HRK_RSA_2048PublicKeyEncrypt()

uint32_t R_SCE_HRK_RSA_2048PublicKeyEncrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt using 2048-bit RSA public key.

Encrypt num_words words of input data from buffer p_source using the 2048-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 64 words of data
num_words must be equal to RSA MODULUS size in bytes (RSA Public Key_size / bits per byte). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/8 = 256 bytes or 64 words.

◆ R_SCE_HRK_RSA_2048PublicKeyVerify()

uint32_t R_SCE_HRK_RSA_2048PublicKeyVerify (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_signature*, uint32_t * *p_padded_hash*)

Signature Verification using 2048-bit RSA public key.

Verify RSA signature data from buffer p_signature of length num_words using 2048-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature or/and Input parameter p_padded_hash is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 64 words.
Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 2048-bit key, num_words should be equal to 64.
num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/32 = 64 words.

◆ R_SCE_RSA_1024KeyCreate()

uint32_t R_SCE_RSA_1024KeyCreate (rsa_ctrl_t *const *p_ctrl*, rsa_key_t * *p_private_key*, rsa_key_t * *p_public_key*)

Creation of 1024-bit RSA Key pair in plain text format.

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY specifies a standard format private key.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY specifies the CRT parameters to be created.

Return values

SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not be empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

Note

The buffers to hold the keys must be adequate for the key formats requested.
The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, in that order.

◆ R_SCE_RSA_1024PrivateCrtKeyDecrypt()

uint32_t R_SCE_RSA_1024PrivateCrtKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 1024-bit RSA private key (CRT format)

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key pointer to 80-word buffer with 1024-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_RSA_1024PrivateCrtKeySign()

uint32_t R_SCE_RSA_1024PrivateCrtKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 1024-bit RSA private key represented in CRT format.

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key pointer to 80-word buffer with 1024-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_RSA_1024PrivateKeyDecrypt()

uint32_t R_SCE_RSA_1024PrivateKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 1024-bit RSA private key (standard format)

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 32 words of RSA private key data followed by 32 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_RSA_1024PrivateKeySign()

uint32_t R_SCE_RSA_1024PrivateKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 1024-bit RSA private key.

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 32 words of RSA private key data followed by 32 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

◆ R_SCE_RSA_1024PublicKeyEncrypt()

uint32_t R_SCE_RSA_1024PublicKeyEncrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encryption using 1024-bit RSA public key.

Encrypt num_words words of input data from buffer p_source using the 1024-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Note

p_dest must have space to hold at least num_words words of data.
The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 32 words of data
num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/32 = 32 words.

◆ R_SCE_RSA_1024PublicKeyVerify()

uint32_t R_SCE_RSA_1024PublicKeyVerify (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *num_words*, uint32_t * *p_signature*, uint32_t * *p_padded_hash*)

Signature Verification using 1024-bit RSA public key.

Verify RSA signature data from buffer p_signature of length num_words using 1024-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Return values

SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature or/and Input parameter p_padded_hash is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 32 words.
Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 1024-bit key, num_words should be equal to 32.
num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Pubic key = 1024/32 = 32 words.

◆ R_SCE_RSA_2048KeyCreate()

uint32_t R_SCE_RSA_2048KeyCreate (rsa_ctrl_t *const *p_ctrl*, rsa_key_t * *p_private_key*, rsa_key_t * *p_public_key*)

Creation of 2048-bit RSA Key pair in plain text format.

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY specifies a standard format private key.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY specifies the CRT parameters to be created.

Return values

SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not be empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

Note

The buffers to hold the keys must be adequate for the key formats requested.
The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, in that order.

◆ R_SCE_RSA_2048PrivateCrtKeyDecrypt()

uint32_t R_SCE_RSA_2048PrivateCrtKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 2048-bit RSA private key (CRT format)

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain pointer to 160-word buffer with 2048-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)
imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

◆ R_SCE_RSA_2048PrivateCrtKeySign()

uint32_t R_SCE_RSA_2048PrivateCrtKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 2048-bit RSA private key represented in CRT format.

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain pointer to 160-word buffer with 2048-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)
imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

◆ R_SCE_RSA_2048PrivateKeyDecrypt()

uint32_t R_SCE_RSA_2048PrivateKeyDecrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Decrypt using 2048-bit RSA private key (standard format)

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

◆ R_SCE_RSA_2048PrivateKeySign()

uint32_t R_SCE_RSA_2048PrivateKeySign (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Signature generation using 2048-bit RSA private key.

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data
imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

◆ R_SCE_RSA_2048PublicKeyEncrypt()

uint32_t R_SCE_RSA_2048PublicKeyEncrypt (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_source*, uint32_t * *p_dest*)

Encrypt using 2048-bit RSA public key.

Encrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty or/and Input parameter is not valid.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open

Note

p_dest must have space to hold at least imaxcnt words of data.
The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 64 words of data
imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/32 = 64 words.

◆ R_SCE_RSA_2048PublicKeyVerify()

uint32_t R_SCE_RSA_2048PublicKeyVerify (rsa_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, const uint32_t * *p_domain*, uint32_t *imaxcnt*, uint32_t * *p_signature*, uint32_t * *p_paddedHash*)

Signature Verification using 2048-bit RSA public key.

Verify RSA signature data from buffer p_signature of length imaxcnt using 2048-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Return values

SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature or/and Input parameter p_padded_hash is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Note

The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 64 words.
Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 2048-bit key, num_words should be equal to 64.
imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/32 = 64 words.

◆ R_SCE_RSA_Close()

uint32_t R_SCE_RSA_Close (rsa_ctrl_t *const *p_ctrl*)

Close RSA driver

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.

◆ R_SCE_RSA_Open()

uint32_t R_SCE_RSA_Open (rsa_ctrl_t *const *p_ctrl*, rsa_cfg_t const *const *p_cfg*)

RSA Initialization

Return values

SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Precondition

SCE module must have been initialized by calling crypto_api_t::open.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_RSA_VersionGet()

uint32_t R_SCE_RSA_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_CRYPTO_SUCCESS	Successful in getting the module version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_rsa1024_on_sce

const rsa_api_t g_rsa1024_on_sce

```
=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_RSA_1024PublicKeyEncrypt,
    .decrypt    = R_SCE_RSA_1024PrivateKeyDecrypt,
    .decryptCrt = R_SCE_RSA_1024PrivateCrtKeyDecrypt,
    .sign       = R_SCE_RSA_1024PrivateKeySign,
    .signCrt    = R_SCE_RSA_1024PrivateCrtKeySign,
    .verify    = R_SCE_RSA_1024PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate = R_SCE_RSA_1024KeyCreate
}
```

SCE/RSA implementation of RSA API.

◆ g_rsa1024_on_sce_hrk

const rsa_api_t g_rsa1024_on_sce_hrk

```
=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_HRK_RSA_1024PublicKeyEncrypt,
    .decrypt    = R_SCE_HRK_RSA_1024PrivateKeyDecrypt,
    .decryptCrt = R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt,
    .sign       = R_SCE_HRK_RSA_1024PrivateKeySign,
    .signCrt    = R_SCE_HRK_RSA_1024PrivateCrtKeySign,
    .verify    = R_SCE_HRK_RSA_1024PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate = R_SCE_HRK_RSA_1024KeyCreate
}
```

SCE/RSA implementation of RSA API.

◆ g_rsa2048_on_sce_hrk

const rsa_api_t g_rsa2048_on_sce_hrk

```
=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_HRK_RSA_2048PublicKeyEncrypt,
    .decrypt    = R_SCE_HRK_RSA_2048PrivateKeyDecrypt,
    .decryptCrt = R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt,
    .sign       = R_SCE_HRK_RSA_2048PrivateKeySign,
    .signCrt    = R_SCE_HRK_RSA_2048PrivateCrtKeySign,
    .verify    = R_SCE_HRK_RSA_2048PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate = R_SCE_HRK_RSA_2048KeyCreate
}
```

SCE/RSA implementation of RSA API.

SCE_TDES

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_TDES_Open (tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg) TDES Initialization. More...
uint32_t	R_SCE_TDES_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_TDES_Close (tdes_ctrl_t *const p_ctrl) Close TDES module. More...

uint32_t	R_SCE_TDES_192CbcEncrypt (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Triple DES Encryption with CBC mode. More...
uint32_t	R_SCE_TDES_192CbcDecrypt (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Triple DES Decryption with CBC mode. More...
uint32_t	R_SCE_TDES_192CtrEncrypt (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Triple DES Encryption with CTR mode. More...
uint32_t	R_SCE_TDES_192EcbEncrypt (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Triple DES Encryption with ECB mode. More...
uint32_t	R_SCE_TDES_192EcbDecrypt (tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest) Triple DES Decryption with ECB mode. Decrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data. More...

Variables

const tdes_api_t	g_tdes192ecb_on_sce

Detailed Description

Primitive cryptographic functions.

Triple DES encryption and decryption functions

Function Documentation

◆ R_SCE_TDES_192CbcDecrypt()

uint32_t R_SCE_TDES_192CbcDecrypt (tdes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Triple DES Decryption with CBC mode.

Decrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SSP_ERR_ASSERTION	One of the input parameters is NULL or invalid.
SF_CRYPTO_SUCCESS	Function returned successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.

Precondition

SCE module must have been initialized.

Note

p_dest must have space to hold at least num_words words of data.
The key buffer must contain 24 bytes of TDES key data and
the iv buffer must have 8 bytes of random data.

◆ R_SCE_TDES_192CbcEncrypt()

uint32_t R_SCE_TDES_192CbcEncrypt (tdes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Triple DES Encryption with CBC mode.

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SSP_ERR_ASSERTION	One of the input parameters is NULL or invalid.
SF_CRYPTO_SUCCESS	Function returned successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.

Precondition

SCE module must have been initialized.

Note

p_dest must have space to hold at least num_words words of data.
The key buffer must contain 24 bytes of TDES key data and
the iv buffer must have 8 bytes of random data.

◆ R_SCE_TDES_192CtrEncrypt()

uint32_t R_SCE_TDES_192CtrEncrypt (tdes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Triple DES Encryption with CTR mode.

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Return values

SSP_ERR_ASSERTION	One of the input parameters is NULL or invalid.
SF_CRYPTO_SUCCESS	Function returned successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.

Precondition

SCE module must have been initialized.

Note

p_dest must have space to hold at least num_words words of data.
The key buffer must contain 24 bytes of TDES key data and
the iv buffer must have 8 bytes of random data.

◆ R_SCE_TDES_192EcbDecrypt()

uint32_t R_SCE_TDES_192EcbDecrypt (tdes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Triple DES Decryption with ECB mode. Decrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SSP_ERR_ASSERTION	One of the input parameters is NULL or invalid.
SF_CRYPTO_SUCCESS	Function returned successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.

Precondition

SCE module must have been initialized.

Note

p_dest must have space to hold at least num_words words of data.
The key buffer must contain 24 bytes of TDES key data and
The contents of iv buffer are ignored in ECB chaining mode.

To prevent compiler warning for unused parameter.

◆ R_SCE_TDES_192EcbEncrypt()

uint32_t R_SCE_TDES_192EcbEncrypt (tdes_ctrl_t *const *p_ctrl*, const uint32_t * *p_key*, uint32_t * *p_iv*, uint32_t *num_words*, uint32_t * *p_source*, uint32_t * *p_dest*)

Triple DES Encryption with ECB mode.

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Return values

SSP_ERR_ASSERTION	One of the input parameters is NULL or invalid.
SF_CRYPTO_SUCCESS	Function returned successfully.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.

Precondition

SCE module must have been initialized.

Note

Block size for TDES is 64-bits (8 bytes or 2 words). Hence num_words must be a multiple of 2.
p_dest must have space to hold at least num_words words of data.
The key buffer must contain 24 bytes of TDES key data.
The contents of iv buffer are ignored in ECB chaining mode.

To prevent compiler warning for unused parameter.

◆ R_SCE_TDES_Close()

uint32_t R_SCE_TDES_Close (tdes_ctrl_t *const *p_ctrl*)

Close TDES module.

Return values

SF_CRYPTO_SUCCESS	Module closed successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

◆ R_SCE_TDES_Open()

uint32_t R_SCE_TDES_Open (tdes_ctrl_t *const *p_ctrl*, tdes_cfg_t const *const *p_cfg*)

TDES Initialization.

Return values

SF_CRYPTO_SUCCESS	Module opened successfully.
SSP_ERR_ASSERTION	One of the input parameter is NULL.
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_TDES_VersionGet()

uint32_t R_SCE_TDES_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SF_CRYPTO_SUCCESS	Returned the driver version successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_tdes192ecb_on_sce

const tdes_api_t g_tdes192ecb_on_sce

```
=
{
    .open       = R_SCE_TDES_Open,
    .encrypt    = R_SCE_TDES_192EcbEncrypt,
    .decrypt    = R_SCE_TDES_192EcbDecrypt,
    .close      = R_SCE_TDES_Close,
    .versionGet = R_SCE_TDES_VersionGet
}
```

SCE/TDES implementation of TDES API.

SCE_TRNG

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Primitive cryptographic functions. More...

Functions

uint32_t	R_SCE_TRNG_Open (trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
uint32_t	R_SCE_TRNG_VersionGet (ssp_version_t *const p_version) Sets driver version based on compile time macros. More...
uint32_t	R_SCE_TRNG_Read (trng_ctrl_t *const p_ctrl, uint32_t *const p_dest, uint32_t nwords)
uint32_t	R_SCE_TRNG_Close (trng_ctrl_t *const p_ctrl)

Variables

const trng_api_t	g_trng_on_sce

Detailed Description

Primitive cryptographic functions.

random number generation functions

Function Documentation

◆ R_SCE_TRNG_Close()

uint32_t R_SCE_TRNG_Close (trng_ctrl_t *const *p_ctrl*)

Close the TRNG interface driver

Return values

SF_CRYPTO_SUCCESS	TRNG module was closed successfully.
SSP_ERR_ASSERTION	NULL input parameter(s).

◆ R_SCE_TRNG_Open()

uint32_t R_SCE_TRNG_Open (trng_ctrl_t *const *p_ctrl*, trng_cfg_t const *const *p_cfg*)

SCE_TRNG example: extern const r_sce_t g_<interface>_on_sce;

Open the TRNG driver for reading random data from the hardware TRNG module TRNG Initialization

Return values

SF_CRYPTO_SUCCESS	random number generation successful
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_NOT_OPEN	Crypto HAL Common module is not Opened.

Get status of Crypto HAL common module

Return error code of Crypto HAL common module is not open

◆ R_SCE_TRNG_Read()

uint32_t R_SCE_TRNG_Read (trng_ctrl_t *const *p_ctrl*, uint32_t *const *p_rngbuf*, uint32_t *nwords*)

Generate nwords of random number words (4-bytes each) and store them in p_rngbuf buffer SCE hardware TRNG module will be used for generating the random data.

Return values

SF_CRYPTO_SUCCESS	random number generation successful
SSP_ERR_CRYPTO_RNG_FATAL_ERROR	HW_SCE_RNG_Read failed to generate 128-bit (16-byte) random number in p_ctrl->nattempts number of attempts.
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty
SSP_ERR_ASSERTION	NULL input parameter(s).

◆ R_SCE_TRNG_VersionGet()

uint32_t R_SCE_TRNG_VersionGet (ssp_version_t *const *p_version*)

Sets driver version based on compile time macros.

Return values

SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

Variable Documentation

◆ g_trng_on_sce

const trng_api_t g_trng_on_sce

```
=
{
    .open       = R_SCE_TRNG_Open,
    .read       = R_SCE_TRNG_Read,
    .close      = R_SCE_TRNG_Close,
    .versionGet = R_SCE_TRNG_VersionGet
}
```

SCE/TRNG implementation of RNG API.

SCE_INTERFACE_GET

Renesas Synergy Software Package Reference » HAL Layer » SCE Module

Get Interface for Crypto HAL modules. More...

Functions

uint32_t R_SCE_InterfaceGet (crypto_interface_get_param_t *const interface_info, void *const p_interface)

Get the HAL interface global object based on the service/algorithm requested. More...

Detailed Description

Get Interface for Crypto HAL modules.

Function Documentation

◆ R_SCE_InterfaceGet()

uint32_t R_SCE_InterfaceGet (crypto_interface_get_param_t *const *interface_info*, void *const *p_interface*)

Get the HAL interface global object based on the service/algorithm requested.

Return values

SSP_SUCCESS	Valid interface has been returned.
SSP_ERR_ASSERTION	One or more invalid input parameters.
SSP_ERR_INVALID_ARGUMENT	Invalid request. Input parameters could not be resolved into a valid HAL interface.

5.2 Board Support Package

Common BSP includes. More...

Modules

	Supported MCUs Supported MCUs in this version of the BSP.
	Common BSP Code Code common to all BSPs.

Detailed Description

Common BSP includes.

The BSP is responsible for getting the MCU from reset to the user application (i.e. the main function). Before reaching the user application the BSP sets up the stacks, heap, clocks, interrupts, and C runtime environment. The BSP is configurable to allow users to modify the process to meet design requirements.

5.2.1 Supported MCUs

Board Support Package

Supported MCUs in this version of the BSP. More...

Modules

	S124 Code that is common to S124 MCUs.
	S128 Code that is common to S128 MCUs.
	S1JA Code that is common to S1JA MCUs.
	S3A1 Code that is common to S3A1 MCUs.
	S3A3 Code that is common to S3A3 MCUs.
	S3A6 Code that is common to S3A6 MCUs.
	S3A7 Code that is common to S3A7 MCUs.
	S5D3 Code that is common to S5D3 MCUs.
	S5D5 Code that is common to S5D5 MCUs.

	S5D9 Code that is common to S5D9 MCUs.
	S7G2 Code that is common to S7G2 MCUs.

Functions

ssp_err_t	R_SSP_VersionGet (ssp_pack_version_t *const p_version) Set SSP version based on compile time macros. More...

Detailed Description

Supported MCUs in this version of the BSP.

The BSP has code specific to a MCU and a board. The code that is specific to a MCU can be shared amongst boards that use the MCU.

Function Documentation

◆ R_SSP_VersionGet()

ssp_err_t R_SSP_VersionGet (ssp_pack_version_t *const *p_version*)

Set SSP version based on compile time macros.

Parameters

[out]	p_version	Memory address to return version information to.

Return values

SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

5.2.1.1 S124

Board Support Package » Supported MCUs

Code that is common to S124 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Enumerations

enum	elc_peripheral_t
enum	elc_event_t

Detailed Description

Code that is common to S124 MCUs.

Implements functions that are common to S124 MCUs.

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t
Sources of event signals to be linked to other peripherals or the CPU1 ***Note*** *This list may change based on device. This list is for S124.*

◆ elc_peripheral_t

enum elc_peripheral_t
Possible peripherals to be linked to event signals

Analog Connections

Board Support Package » Supported MCUs » S124

Enumerations

enum	analog_connect_t {

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5,
FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7,
```

```
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
```

FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
```

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_SET),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501

= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S124

Note

This list may change based on device. This list is for S124.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.

ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.

ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.

ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P	Connect OPAMP2 AMPP to PORT0 P003.

003	
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1	

_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S124

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S124

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void)

	Returns frequency of CPU clock in Hz. More...

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL. If the system clock has failed to start call the unrecoverable error handler. MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler.

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency — of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S124

Functions

	SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)
	SSP_HW_LOCK_DEFINE (AES, 0U, 0U)
	SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)
	SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)

SSP_HW_LOCK_DEFINE (TRNG, 0U, 0U)

	SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
	SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
	SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/31]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/31]

SSP_HW_LOCK_DEFINE (AES , 0U , 0U)
AES

◆ SSP_HW_LOCK_DEFINE() [3/31]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [4/31]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [5/31]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [6/31]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [7/31]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [8/31]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [9/31]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [10/31]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [11/31]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/31]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/31]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [14/31]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [15/31]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [16/31]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [17/31]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [18/31]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [19/31]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [20/31]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [21/31]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [22/31]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [23/31]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [24/31]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [25/31]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [26/31]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [27/31]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [28/31]

SSP_HW_LOCK_DEFINE (TRNG , 0U , 0U)
TRNG

◆ SSP_HW_LOCK_DEFINE() [29/31]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [30/31]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [31/31]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S124

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...

ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

```
#define BSP_COMPILE_TIME_ASSERT ( e) ((void) sizeof(char[1 - 2 * !(e)]))
```

Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)
The g_bsp_module_stop array must have entries for each ssp_ip_t enum value. Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)
Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. ***Note*** *Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.* **Parameters** [in] p_feature Pointer to definition of the feature, defined by ssp_feature_t. **Return values** SSP_SUCCESS Module is stopped SSP_ERR_ASSERTION p_feature::id is invalid SSP_ERR_INVALID_ARGUMENT Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)
Stop module (enter module stop) even if the module is used for multiple channels. **Parameters** [in] p_feature Pointer to definition of the feature, defined by ssp_feature_t. **Return values** SSP_SUCCESS Module is stopped SSP_ERR_ASSERTION p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S124

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ **BSP_ROM_REG_OFS1_SETTING**

```
#define BSP_ROM_REG_OFS1_SETTING   (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
```

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.2 S128

Board Support Package » Supported MCUs

Code that is common to S128 MCUs. More...

Modules

Analog Connections
Cache Functions
Clock Initialization
Hardware Locks
Module Start and Stop
ROM Registers

Detailed Description

Code that is common to S128 MCUs.

Implements functions that are common to S128 MCUs.

Analog Connections

Board Support Package » Supported MCUs » S128

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,

FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 =

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),

```
 ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
```

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
```

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_SET),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 =

ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2,

FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA =

```
                                    ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3,
                                    FLAG_CLEAR)
                                    }
```

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S128

Note

This list may change based on device. This list is for S128.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.

ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.

ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.

ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S128

Enumerations

enum bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S128

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)

Sets up system clocks.

MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.

If the system clock has failed to start call the unrecoverable error handler.

MOCO, LOCO, and subclock do not have stabilization flags that can be checked.

Wait for clock source to stabilize

Set which clock to use for system clock and divisors for all system clocks.

If the system clock has failed to be configured properly call the unrecoverable error handler.

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)

Returns frequency of CPU clock in Hz.

Return values

Frequency	of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S128

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AES, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 1U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (TRNG, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/32]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/32]

SSP_HW_LOCK_DEFINE (AES , 0U , 0U)
AES

◆ SSP_HW_LOCK_DEFINE() [3/32]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [4/32]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [5/32]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [6/32]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [7/32]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [8/32]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [9/32]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [10/32]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [11/32]

SSP_HW_LOCK_DEFINE (DAC , 1U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/32]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/32]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [14/32]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [15/32]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [16/32]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [17/32]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [18/32]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [19/32]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [20/32]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [21/32]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [22/32]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [23/32]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [24/32]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [25/32]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [26/32]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [27/32]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [28/32]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [29/32]

SSP_HW_LOCK_DEFINE (TRNG , 0U , 0U)
TRNG

◆ SSP_HW_LOCK_DEFINE() [30/32]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [31/32]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [32/32]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S128

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S128

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.3 S1JA

Board Support Package » Supported MCUs

Code that is common to S1JA MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S1JA MCUs.

Implements functions that are common to S1JA MCUs.

Analog Connections

Board Support Package » Supported MCUs » S1JA

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015

= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
```

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S1JA

Note
This list may change based on device. This list is for S1JA.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.

ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5	Connect ACMPLP1 IVREF1 to PORT5 P500.

_P500	
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_	

P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S1JA

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S1JA

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_mem_wait_set (uint32_t setting) This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. More...
__STATIC_INLINE uint32_t	bsp_clocks_mem_wait_get (void) This function gets the value of the MEMWAIT register. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. S1JA has no PLL. If the system clock has failed to start call the unrecoverable error handler. MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler.

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0. < No MEMWAIT cycles

◆ bsp_clocks_mem_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_mem_wait_get (void)
This function gets the value of the MEMWAIT register. **Return values** MEMWAIT: setting

◆ bsp_clocks_mem_wait_set()

__STATIC_INLINE void bsp_clocks_mem_wait_set (uint32_t *setting*)
This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Parameters

[in]	setting	

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz.

Return values

Frequency	of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S1JA

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AES, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SDADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (TRNG, 0U, 0U)

SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)

SSP_HW_LOCK_DEFINE (USB, 0U, 0U)

SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/33]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/33]

SSP_HW_LOCK_DEFINE (AES , 0U , 0U)
AES

◆ SSP_HW_LOCK_DEFINE() [3/33]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [4/33]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [5/33]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [6/33]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [7/33]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [8/33]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [9/33]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [10/33]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [11/33]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/33]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/33]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [14/33]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [15/33]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [16/33]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [17/33]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [18/33]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [19/33]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [20/33]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [21/33]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [22/33]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [23/33]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [24/33]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [25/33]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [26/33]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [27/33]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [28/33]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [29/33]

SSP_HW_LOCK_DEFINE (SDADC , 0U , 0U)
SDADC

◆ SSP_HW_LOCK_DEFINE() [30/33]

SSP_HW_LOCK_DEFINE (TRNG , 0U , 0U)
TRNG

◆ SSP_HW_LOCK_DEFINE() [31/33]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [32/33]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [33/33]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S1JA

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

```
#define BSP_COMPILE_TIME_ASSERT ( e) ((void) sizeof(char[1 - 2 * !(e)]))
```

Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S1JA

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.4 S3A1

Board Support Package » Supported MCUs

Code that is common to S3A1 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Enumerations

enum	elc_peripheral_t
enum	elc_event_t

Detailed Description

Code that is common to S3A1 MCUs.

Implements functions that are common to S3A1 MCUs.

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1

Note
This list may change based on device. This list is for S3A7.

◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

Analog Connections

Board Support Package » Supported MCUs » S3A1

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5,
FLAG_SET),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
```

```
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 =
```

ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
```

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
```

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
 ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_SET),
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
```

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA

= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3,
FLAG_CLEAR)
}
```

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S3A1

Note

This list may change based on device. This list is for S3A1.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80	Connect ACMPLP0 IVREF0 to DAC80 DA.

_DA	
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1	

_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P	Connect OPAMP0 AMPM to PORT0 P014.

014	
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.

ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.

ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S3A1

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S3A1

Functions

void	bsp_clock_init (void) Sets up system clocks. More...

uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_mem_wait_set (uint32_t setting) This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. More...
__STATIC_INLINE uint32_t	bsp_clocks_mem_wait_get (void) This function gets the value of the MEMWAIT register. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. PLL Source clock is always the main oscillator. Need to start PLL source clock and let it stabilize before starting PLL Set PLL Divider. Set PLL Multiplier. PLL Source clock is always the main oscillator. Wait for PLL clock source to stabilize If the system clock has failed to start call the unrecoverable error handler. MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler. If the USB clock source requested is HOCO set the corresponding bit in the USBCKCR register

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0. < No MEMWAIT cycles

◆ bsp_clocks_mem_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_mem_wait_get (void)
This function gets the value of the MEMWAIT register. **Return values** MEMWAIT: setting

◆ bsp_clocks_mem_wait_set()

__STATIC_INLINE void bsp_clocks_mem_wait_set (uint32_t *setting*)
This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. **Parameters** [in] setting **Return values** none

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency: of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S3A1

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)
SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)
SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)
SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)
SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SLCDC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/38]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/38]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/38]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/38]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/38]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/38]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [7/38]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [8/38]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [9/38]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [10/38]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [11/38]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [12/38]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [13/38]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [14/38]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [15/38]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [16/38]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [17/38]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [18/38]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [19/38]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [20/38]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [21/38]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [22/38]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [23/38]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [24/38]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [25/38]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [26/38]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [27/38]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [28/38]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [29/38]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [30/38]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [31/38]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [32/38]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [33/38]

SSP_HW_LOCK_DEFINE (SLCDC , 0U , 0U)
SLCDC

◆ SSP_HW_LOCK_DEFINE() [34/38]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [35/38]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [36/38]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [37/38]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [38/38]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S3A1

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)
Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)
The g_bsp_module_stop array must have entries for each ssp_ip_t enum value. Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S3A1

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.5 S3A3

Board Support Package » Supported MCUs

Code that is common to S3A3 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S3A3 MCUs.

Implements functions that are common to S3A3 MCUs.

Analog Connections

Board Support Package » Supported MCUs » S3A3

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015

= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,

```
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP2_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK,
FLAG_CLEAR),
  ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
```

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
```

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t	
List of analog connections that can be made on S3A3 ***Note*** *This list may change based on device. This list is for S3A3.*	
Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.

ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5	Connect ACMPLP1 IVREF1 to PORT5 P500.

_P500	
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_	

P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S3A3

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S3A3

Macros

#define	CGC_SRAM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for SRAM.
#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_mem_wait_set (uint32_t setting) This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. More...

__STATIC_INLINE uint32_t	bsp_clocks_mem_wait_get (void) This function gets the value of the MEMWAIT register. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)

Sets up system clocks.

MOCO is default clock out of reset. Enable new clock if chosen.

PLL Source clock is always the main oscillator.

Need to start PLL source clock and let it stabilize before starting PLL

Set PLL Divider.

Set PLL Multiplier.

PLL Source clock is always the main oscillator.

Wait for PLL clock source to stabilize

If the system clock has failed to start call the unrecoverable error handler.

Enable ROM cache

MOCO, LOCO, and subclock do not have stabilization flags that can be checked.

Wait for clock source to stabilize

Set which clock to use for system clock and divisors for all system clocks.

If the system clock has failed to be configured properly call the unrecoverable error handler.

If the USB clock source requested is HOCO set the corresponding bit in the USBCKCR register

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

◆ bsp_clocks_mem_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_mem_wait_get (void)
This function gets the value of the MEMWAIT register. **Return values** MEMWAIT: setting

◆ bsp_clocks_mem_wait_set()

__STATIC_INLINE void bsp_clocks_mem_wait_set (uint32_t *setting*)
This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. **Return values** none

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency: of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S3A3

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SLCDC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/40]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/40]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/40]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/40]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/40]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/40]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/40]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [8/40]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [9/40]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [10/40]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [11/40]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/40]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/40]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [14/40]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/40]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/40]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [17/40]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [18/40]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [19/40]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [20/40]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [21/40]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [22/40]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [23/40]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [24/40]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [25/40]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [26/40]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [27/40]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [28/40]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [29/40]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [30/40]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [31/40]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [32/40]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [33/40]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [34/40]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [35/40]

SSP_HW_LOCK_DEFINE (SLCDC , 0U , 0U)
SLCDC

◆ SSP_HW_LOCK_DEFINE() [36/40]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [37/40]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [38/40]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [39/40]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [40/40]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S3A3

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)
Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)
The g_bsp_module_stop array must have entries for each ssp_ip_t enum value. Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S3A3

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3)   \
                                    )
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.6 S3A6

Board Support Package » Supported MCUs

Code that is common to S3A6 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S3A6 MCUs.

Implements functions that are common to S3A6 MCUs.

Analog Connections

Board Support Package » Supported MCUs » S3A6

Enumerations

enum	analog_connect_t { ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,

```
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1,
```

FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),

```
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
```

```
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
 ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
```

FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500

```
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3,
FLAG_CLEAR)
}
```

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S3A6

Note
This list may change based on device. This list is for S3A6.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_	Connect ACMPLP1 IVREF to PORT1 P103.

P103	
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALO	

G0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.

ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_	

DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.

ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.

ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S3A6

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S3A6

Macros

#define	CGC_SRAM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for SRAM.
#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_mem_wait_set (uint32_t setting) This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. More...

__STATIC_INLINE uint32_t	bsp_clocks_mem_wait_get (void) This function gets the value of the MEMWAIT register. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)

Sets up system clocks.

MOCO is default clock out of reset. Enable new clock if chosen.

PLL Source clock is always the main oscillator.

Need to start PLL source clock and let it stabilize before starting PLL

Set PLL Divider.

Set PLL Multiplier.

PLL Source clock is always the main oscillator.

Wait for PLL clock source to stabilize

If the system clock has failed to start call the unrecoverable error handler.

Enable ROM cache

MOCO, LOCO, and subclock do not have stabilization flags that can be checked.

Wait for clock source to stabilize

Set which clock to use for system clock and divisors for all system clocks.

If the system clock has failed to be configured properly call the unrecoverable error handler.

If the USB clock source requested is HOCO set the corresponding bit in the USBCKCR register

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

◆ bsp_clocks_mem_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_mem_wait_get (void)
This function gets the value of the MEMWAIT register. **Return values** MEMWAIT: setting

◆ bsp_clocks_mem_wait_set()

__STATIC_INLINE void bsp_clocks_mem_wait_set (uint32_t *setting*)
This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. **Return values** none

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency: of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S3A6

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SLCDC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/40]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/40]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/40]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/40]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/40]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/40]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/40]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [8/40]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [9/40]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [10/40]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [11/40]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/40]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/40]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [14/40]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/40]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/40]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [17/40]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [18/40]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [19/40]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [20/40]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [21/40]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [22/40]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [23/40]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [24/40]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [25/40]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [26/40]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [27/40]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [28/40]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [29/40]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [30/40]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [31/40]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [32/40]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [33/40]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [34/40]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [35/40]

SSP_HW_LOCK_DEFINE (SLCDC , 0U , 0U)
SLCDC

◆ SSP_HW_LOCK_DEFINE() [36/40]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [37/40]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [38/40]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [39/40]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [40/40]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S3A6

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)
Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)
The g_bsp_module_stop array must have entries for each ssp_ip_t enum value. Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S3A6

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3)   \
                                    )
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.7 S3A7

Board Support Package » Supported MCUs

Code that is common to S3A7 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Enumerations

enum	elc_peripheral_t
enum	elc_event_t

Detailed Description

Code that is common to S3A7 MCUs.

Implements functions that are common to S3A7 MCUs.

Enumeration Type Documentation

◆ **elc_event_t**

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1

Note

This list may change based on device. This list is for S3A7.

◆ **elc_peripheral_t**

enum elc_peripheral_t

Possible peripherals to be linked to event signals

Analog Connections

Board Support Package » Supported MCUs » S3A7

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO =

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
```

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =

ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_SET),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101

= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =

ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502

= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA

= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S3A7

Note

This list may change based on device. This list is for S3A7.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80	Connect ACMPLP0 IVREF0 to DAC80 DA.

_DA	
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1	

_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P	Connect OPAMP0 AMPM to PORT0 P014.

014	
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.

ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.

ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S3A7

Enumerations

enum bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S3A7

Macros

#define CGC_SRAM_ZERO_WAIT_CYCLES (0U)

Specify zero wait states for SRAM.

#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_mem_wait_set (uint32_t setting) This function sets the value of the MEMWAIT register which controls wait cycles for flash read access. More...
__STATIC_INLINE uint32_t	bsp_clocks_mem_wait_get (void) This function gets the value of the MEMWAIT register. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. PLL Source clock is always the main oscillator. Need to start PLL source clock and let it stabilize before starting PLL Set PLL Divider. Set PLL Multiplier. PLL Source clock is always the main oscillator. Wait for PLL clock source to stabilize If the system clock has failed to start call the unrecoverable error handler. MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler.

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0. < No MEMWAIT cycles

◆ bsp_clocks_mem_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_mem_wait_get (void)
This function gets the value of the MEMWAIT register. **Return values** MEMWAIT — setting

◆ bsp_clocks_mem_wait_set()

__STATIC_INLINE void bsp_clocks_mem_wait_set (uint32_t *setting*)
This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Parameters

[in]	setting	

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz.

Return values

Frequency	of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S3A7

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_LP, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPAMP, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)

	SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
	SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
	SSP_HW_LOCK_DEFINE (SLCDC, 0U, 0U)
	SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
	SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
	SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
	SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
	SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/40]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)

Used to allocated hardware locks. Parameters are as follows:

1. IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/40]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)

AGT

◆ SSP_HW_LOCK_DEFINE() [3/40]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)

BSC

◆ SSP_HW_LOCK_DEFINE() [4/40]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/40]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/40]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/40]

SSP_HW_LOCK_DEFINE (COMP_LP , 0U , 0U)
COMP_LP

◆ SSP_HW_LOCK_DEFINE() [8/40]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [9/40]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [10/40]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [11/40]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [12/40]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [13/40]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [14/40]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/40]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/40]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [17/40]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [18/40]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [19/40]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [20/40]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [21/40]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [22/40]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [23/40]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [24/40]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [25/40]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [26/40]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [27/40]

SSP_HW_LOCK_DEFINE (OPAMP , 0U , 0U)
OPAMP

◆ SSP_HW_LOCK_DEFINE() [28/40]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [29/40]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [30/40]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [31/40]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [32/40]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [33/40]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [34/40]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [35/40]

SSP_HW_LOCK_DEFINE (SLCDC , 0U , 0U)
SLCDC

◆ SSP_HW_LOCK_DEFINE() [36/40]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [37/40]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [38/40]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [39/40]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [40/40]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S3A7

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)
Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)
The g_bsp_module_stop array must have entries for each ssp_ip_t enum value. Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S3A7

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCFEU) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                        ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.8 S5D3

Board Support Package » Supported MCUs

Code that is common to S5D3 MCUs. More...

Modules

Analog Connections

	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S5D3 MCUs.

Implements functions that are common to S5D3 MCUs.

Analog Connections
Board Support Package » Supported MCUs » S5D3

Enumerations

enum	analog_connect_t { ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001

= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0,

FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100

= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_SET),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 =

ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4
```

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502

= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA

= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S5D3

Note
This list may change based on device. This list is for S5D3.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0	Connect ACMPHS1 IVCMP to PORT0 P015.

_P015	
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0	

_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.

ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.

ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.

ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S5D3

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S5D3

Macros

#define	CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS (60000000U)
#define	CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS (40000000U)
#define	CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS (80000000U)
#define	CGC_SRAM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for SRAM.
#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_rom_wait_set (uint8_t setting) This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM. More...
__STATIC_INLINE uint32_t	bsp_clocks_rom_wait_get (void) This function gets the value of the ROMWT register. More...

__STATIC_INLINE void	bsp_clocks_sram_wait_set (uint32_t setting) This function sets the RAM wait state settings for both the SRAM0 and SRAM0(ECC) RAM memory. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Macro Definition Documentation

◆ CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS   (60000000U)
```

SRAM requires 1 wait state be inserted at ICLK > 60 MHz. SRAMHS is always no wait state

◆ CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS   (40000000U)
```

FLASH requires 1 wait state be inserted when (40 MHz < ICLK <= 80 MHz)

◆ CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS   (80000000U)
```

FLASH requires 2 wait states be inserted when (80 MHz < ICLK <= 120 MHz)

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. Need to start PLL source clock and let it stabilize before starting PLL Set PLL Divider. Set PLL Multiplier. Set PLL Source clock. Wait for PLL clock source to stabilize If the system clock has failed to start call the unrecoverable error handler. MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler. Set USB clock divisor. Configure BCLK Configure SDRAM Clock

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
Wait states for low speed RAM (SRAM0 and SRAM1) No wait: ICLK <= 60 MHz 1 wait: ICLK > 60 MHz Calculate the Wait states for ROM Set the wait state BEFORE we change iclk In this case we need to set the wait state AFTER we change iclk

◆ bsp_clocks_rom_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_rom_wait_get (void)

This function gets the value of the ROMWT register.

Return values

MEMWAIT	setting

◆ bsp_clocks_rom_wait_set()

__STATIC_INLINE void bsp_clocks_rom_wait_set (uint8_t *setting*)

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_clocks_sram_wait_set()

__STATIC_INLINE void bsp_clocks_sram_wait_set (uint32_t *setting*)

This function sets the RAM wait state settings for both the SRAM0 and SRAM0(ECC) RAM memory.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency — of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S5D3

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)
SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)
SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)
SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)
SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)
SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)
SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)
SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)
SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SRC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/35]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/35]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/35]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [4/35]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [5/35]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [6/35]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [7/35]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [8/35]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [9/35]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [10/35]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [11/35]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [12/35]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [13/35]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [14/35]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [15/35]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [16/35]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [17/35]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [18/35]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [19/35]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [20/35]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [21/35]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [22/35]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [23/35]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [24/35]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [25/35]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [26/35]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [27/35]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [28/35]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [29/35]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [30/35]

SSP_HW_LOCK_DEFINE (SRC , 0U , 0U)
SRC

◆ SSP_HW_LOCK_DEFINE() [31/35]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [32/35]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [33/35]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [34/35]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [35/35]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S5D3

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S5D3

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.9 S5D5

Board Support Package » Supported MCUs

Code that is common to S5D5 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S5D5 MCUs.

Implements functions that are common to S5D5 MCUs.

Analog Connections

Board Support Package » Supported MCUs » S5D5

Enumerations

enum	analog_connect_t { ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015

= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
```

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_SET),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_SET),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =

ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S5D5

Note
This list may change based on device. This list is for S5D5.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.

ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5	Connect ACMPLP1 IVREF1 to PORT5 P500.

_P500	
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_	

P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S5D5

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S5D5

Macros

#define	CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS (60000000U)
#define	CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS (40000000U)
#define	CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS (80000000U)
#define	CGC_SRAM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for SRAM.
#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...

__STATIC_INLINE void	bsp_clocks_rom_wait_set (uint8_t setting) This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM. More...
__STATIC_INLINE uint32_t	bsp_clocks_rom_wait_get (void) This function gets the value of the ROMWT register. More...
__STATIC_INLINE void	bsp_clocks_sram_wait_set (uint32_t setting) This function sets the RAM wait state settings for both the SRAM0 and SRAM0(ECC) RAM memory. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Macro Definition Documentation

◆ CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS   (60000000U)
```

SRAM requires 1 wait state be inserted at ICLK > 60 MHz. SRAMHS is always no wait state

◆ CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS   (40000000U)
```

FLASH requires 1 wait state be inserted when (40 MHz < ICLK <= 80 MHz)

◆ CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS   (80000000U)
```

FLASH requires 2 wait states be inserted when (80 MHz < ICLK <= 120 MHz)

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. Need to start PLL source clock and let it stabilize before starting PLL Set PLL Divider. Set PLL Multiplier. Set PLL Source clock. Wait for PLL clock source to stabilize If the system clock has failed to start call the unrecoverable error handler. Enable ROM cache MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler. Set USB clock divisor. Configure BCLK Configure SDRAM Clock

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
Wait states for low speed RAM (SRAM0 and SRAM1) No wait: ICLK <= 60 MHz 1 wait: ICLK > 60 MHz Calculate the Wait states for ROM Set the wait state BEFORE we change iclk In this case we need to set the wait state AFTER we change iclk

◆ bsp_clocks_rom_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_rom_wait_get (void)

This function gets the value of the ROMWT register.

Return values

MEMWAIT	setting

◆ bsp_clocks_rom_wait_set()

__STATIC_INLINE void bsp_clocks_rom_wait_set (uint8_t *setting*)

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_clocks_sram_wait_set()

__STATIC_INLINE void bsp_clocks_sram_wait_set (uint32_t *setting*)

This function sets the RAM wait state settings for both the SRAM0 and SRAM0(ECC) RAM memory.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency — of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S5D5

Functions

- SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)
- SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)
- SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)
- SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)
- SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DRW, 0U, 0U)
- SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)
- SSP_HW_LOCK_DEFINE (EPTPC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ETHER, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GLCDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (JPEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (PDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)

	SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
	SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
	SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
	SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/44]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/44]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/44]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/44]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/44]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/44]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/44]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [8/44]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [9/44]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [10/44]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [11/44]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [12/44]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [13/44]

SSP_HW_LOCK_DEFINE (DRW , 0U , 0U)
DRW

◆ SSP_HW_LOCK_DEFINE() [14/44]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/44]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/44]

SSP_HW_LOCK_DEFINE (EPTPC , 0U , 0U)
EPTPC

◆ SSP_HW_LOCK_DEFINE() [17/44]

SSP_HW_LOCK_DEFINE (ETHER , 0U , 0U)
ETHER

◆ SSP_HW_LOCK_DEFINE() [18/44]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [19/44]

SSP_HW_LOCK_DEFINE (GLCDC , 0U , 0U)
GLCDC

◆ SSP_HW_LOCK_DEFINE() [20/44]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [21/44]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [22/44]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [23/44]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [24/44]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [25/44]

SSP_HW_LOCK_DEFINE (JPEG , 0U , 0U)
JPEG

◆ SSP_HW_LOCK_DEFINE() [26/44]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [27/44]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [28/44]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [29/44]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [30/44]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [31/44]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [32/44]

SSP_HW_LOCK_DEFINE (PDC , 0U , 0U)
PDC

◆ SSP_HW_LOCK_DEFINE() [33/44]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [34/44]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [35/44]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [36/44]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [37/44]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [38/44]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [39/44]

SSP_HW_LOCK_DEFINE (SRC , 0U , 0U)
SRC

◆ SSP_HW_LOCK_DEFINE() [40/44]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [41/44]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [42/44]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [43/44]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [44/44]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S5D5

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature)

	Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

```
#define BSP_COMPILE_TIME_ASSERT ( e) ((void) sizeof(char[1 - 2 * !(e)]))
```

Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S5D5

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.10 S5D9

Board Support Package » Supported MCUs

Code that is common to S5D9 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Detailed Description

Code that is common to S5D9 MCUs.

Implements functions that are common to S5D9 MCUs.

Analog Connections

Board Support Package » Supported MCUs » S5D9

Enumerations

enum	analog_connect_t { ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015

= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5, FLAG_SET),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,

FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3, FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK, FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1, FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
```

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_SET),
  ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
```

ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =

ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501

= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR)
}

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t

List of analog connections that can be made on S5D9

Note
This list may change based on device. This list is for S5D9.

Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014	Connect OPAMP0 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.

ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.
ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5	Connect ACMPLP1 IVREF1 to PORT5 P500.

_P500	
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_	

P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.

ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S5D9

Enumerations

enum	bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t
Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S5D9

Macros

#define	CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS (60000000U)
#define	CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS (40000000U)
#define	CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS (80000000U)
#define	CGC_SRAM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for SRAM.
#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...

__STATIC_INLINE void	bsp_clocks_rom_wait_set (uint8_t setting) This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM. More...
__STATIC_INLINE uint32_t	bsp_clocks_rom_wait_get (void) This function gets the value of the ROMWT register. More...
__STATIC_INLINE void	bsp_clocks_sram_wait_set (uint32_t setting) This function sets the RAM wait state settings for the SRAM0, SRAM0 ECC and SRAM1 RAM memory. More...
ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args)

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Macro Definition Documentation

◆ CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS   (60000000U)
```

SRAM requires 1 wait state be inserted at ICLK > 60 MHz. SRAMHS is always no wait state

◆ CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS   (40000000U)
```

FLASH requires 1 wait state be inserted when (40 MHz < ICLK <= 80 MHz)

◆ CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

```
#define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS   (80000000U)
```

FLASH requires 2 wait states be inserted when (80 MHz < ICLK <= 120 MHz)

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)
Sets up system clocks. MOCO is default clock out of reset. Enable new clock if chosen. Need to start PLL source clock and let it stabilize before starting PLL Set PLL Divider. Set PLL Multiplier. Set PLL Source clock. Wait for PLL clock source to stabilize If the system clock has failed to start call the unrecoverable error handler. Enable ROM cache MOCO, LOCO, and subclock do not have stabilization flags that can be checked. Wait for clock source to stabilize Set which clock to use for system clock and divisors for all system clocks. If the system clock has failed to be configured properly call the unrecoverable error handler. Set USB clock divisor. Configure BCLK Configure SDRAM Clock

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)
Wait states for low speed RAM (SRAM0 and SRAM1) No wait: ICLK <= 60 MHz 1 wait: ICLK > 60 MHz Calculate the Wait states for ROM Set the wait state BEFORE we change iclk In this case we need to set the wait state AFTER we change iclk

◆ bsp_clocks_rom_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_rom_wait_get (void)

This function gets the value of the ROMWT register.

Return values

MEMWAIT	setting

◆ bsp_clocks_rom_wait_set()

__STATIC_INLINE void bsp_clocks_rom_wait_set (uint8_t *setting*)

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_clocks_sram_wait_set()

__STATIC_INLINE void bsp_clocks_sram_wait_set (uint32_t *setting*)

This function sets the RAM wait state settings for the SRAM0, SRAM0 ECC and SRAM1 RAM memory.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)
Returns frequency of CPU clock in Hz. **Return values** Frequency — of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S5D9

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DRW, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (EPTPC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ETHER, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GLCDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (JPEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (PDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)

SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)

SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)

SSP_HW_LOCK_DEFINE (SRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)

	SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
	SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
	SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
	SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/44]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/44]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/44]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/44]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/44]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/44]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/44]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [8/44]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [9/44]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [10/44]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [11/44]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [12/44]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [13/44]

SSP_HW_LOCK_DEFINE (DRW , 0U , 0U)
DRW

◆ SSP_HW_LOCK_DEFINE() [14/44]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/44]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/44]

SSP_HW_LOCK_DEFINE (EPTPC , 0U , 0U)
EPTPC

◆ SSP_HW_LOCK_DEFINE() [17/44]

SSP_HW_LOCK_DEFINE (ETHER , 0U , 0U)
ETHER

◆ SSP_HW_LOCK_DEFINE() [18/44]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [19/44]

SSP_HW_LOCK_DEFINE (GLCDC , 0U , 0U)
GLCDC

◆ SSP_HW_LOCK_DEFINE() [20/44]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [21/44]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [22/44]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [23/44]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [24/44]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [25/44]

SSP_HW_LOCK_DEFINE (JPEG , 0U , 0U)
JPEG

◆ SSP_HW_LOCK_DEFINE() [26/44]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [27/44]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [28/44]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [29/44]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [30/44]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [31/44]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [32/44]

SSP_HW_LOCK_DEFINE (PDC , 0U , 0U)
PDC

◆ SSP_HW_LOCK_DEFINE() [33/44]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [34/44]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [35/44]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [36/44]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [37/44]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [38/44]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [39/44]

SSP_HW_LOCK_DEFINE (SRC , 0U , 0U)
SRC

◆ SSP_HW_LOCK_DEFINE() [40/44]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [41/44]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [42/44]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [43/44]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [44/44]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S5D9

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature)

	Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

```
#define BSP_COMPILE_TIME_ASSERT ( e) ((void) sizeof(char[1 - 2 * !(e)]))
```

Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S5D9

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))
#define	BSP_ROM_REG_MPU_CONTROL_SETTING

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_MPU_CONTROL_SETTING

#define BSP_ROM_REG_MPU_CONTROL_SETTING

```
((0xFFFFFCF0U) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABL
E << 8) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABL
E << 9) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_E
NABLE) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION1_E
NABLE << 1) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION2_E
NABLE << 2) | \
                                       ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION3_E
NABLE << 3))
```

Build up SECMPUAC register based on MPU settings.

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))

OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.1.11 S7G2

Board Support Package » Supported MCUs

Code that is common to S7G2 MCUs. More...

Modules

	Analog Connections
	Cache Functions
	Clock Initialization
	Hardware Locks
	Module Start and Stop
	ROM Registers

Enumerations

enum	elc_peripheral_t
enum	elc_event_t

Detailed Description

Code that is common to S7G2 MCUs.

Implements functions that are common to S7G2 MCUs.

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1

Note

This list may change based on device. This list is for S7G2.

◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

Analog Connections

Board Support Package » Supported MCUs » S7G2

Enumerations

enum analog_connect_t {
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5, FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO = ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,

```
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF5,
FLAG_SET),
  ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P109 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
  ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
```

```
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0OS, AMPOS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS4,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014 =
```

```
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS1,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP0PS, AMPPS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS0,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS2,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS3,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP1PS, AMPPS7,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPM_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, BREAK,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS0,
FLAG_CLEAR),
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2MS, AMPMS7,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_BREAK =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, BREAK,
FLAG_CLEAR),
 ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS0,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS1,
FLAG_CLEAR), ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA =
ANALOG_CONNECT_DEFINE(OPAMP, 0, AMP2PS, AMPPS7,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
```

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS5,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CRVS6,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL0,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL4,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL0, CMPSEL6,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, CLEAR_C1VRF2,

```
FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1 =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPSEL1, C1VRF2,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_SET),
 ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_SET),
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP5,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF4,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF5,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C0VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C0VRF,
```

FLAG_CLEAR),
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, C1VRF,
FLAG_CLEAR), ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103
= ANALOG_CONNECT_DEFINE(ACMPLP, 0, COMPMDR, CLEAR_C1VRF,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2
= ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3,
FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 =
ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR),

ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF =

ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0,

FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA

= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP1,

FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 0, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 1, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP2, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL0, IVCMP3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501 = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF1, FLAG_CLEAR),
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF2, FLAG_SET), ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 2, CMPSEL1, IVREF3, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502 = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP0, FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA = ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP1, FLAG_CLEAR),

```
 ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004 =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 3, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005 =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 4, CMPSEL1, IVREF3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006 =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP2,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL0, IVCMP3,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF0,
FLAG_CLEAR), ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501
= ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF1,
FLAG_CLEAR),
 ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF2,
FLAG_SET), ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA =
ANALOG_CONNECT_DEFINE(ACMPHS, 5, CMPSEL1, IVREF3,
FLAG_CLEAR)
}
```

Detailed Description

This group contains a list of enumerations that can be used with the Analog Connect Interface.

Enumeration Type Documentation

◆ analog_connect_t

enum analog_connect_t	
List of analog connections that can be made on S7G2 ***Note*** *This list may change based on device. This list is for S7G2.*	
Enumerator	
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P012	Connect ACMPHS0 IVREF to PORT0 P012.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC80_DA	Connect ACMPHS1 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P000	Connect ACMPHS2 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT0_P001	Connect ACMPHS2 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC80_DA	Connect ACMPHS2 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC82_DA	Connect ACMPHS2 IVREF to DAC82 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80	Connect ACMPLP0 IVREF0 to DAC80 DA.

_DA	
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP0 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP2_AMPO	Connect ACMPLP1 IVCMP to OPAMP2 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P500	Connect ACMPHS0 IVCMP to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P013	Connect ACMPHS0 IVCMP to PORT0 P013.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT1_P100	Connect ACMPHS0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT1_P101	Connect ACMPHS0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC80_DA	Connect ACMPHS0 IVREF to DAC80 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1	

_P109	Connect ACMPLP0 IVREF0 to PORT1 P109.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P110	Connect ACMPLP1 IVREF1 to PORT1 P110.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT4_P400	Connect ACMPLP0 IVCMP to PORT4 P400.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_OPAMP0_AMPO	Connect ACMPLP0 IVCMP to OPAMP0 AMPO.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT4_P408	Connect ACMPLP1 IVCMP to PORT4 P408.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_OPAMP1_AMPO	Connect ACMPLP1 IVCMP to OPAMP1 AMPO.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_OPAMP0_AMPO_BREAK	Break all connections to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P014	Connect OPAMP0 AMPO to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P013	Connect OPAMP0 AMPO to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P003	Connect OPAMP0 AMPO to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPO_TO_PORT0_P002	Connect OPAMP0 AMPO to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPM_BREAK	Break all connections to OPAMP0 AMPM.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P501	Connect OPAMP0 AMPM to PORT5 P501.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT5_P500	Connect OPAMP0 AMPM to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P	Connect OPAMP0 AMPM to PORT0 P014.

014	
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P013	Connect OPAMP0 AMPM to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPM_TO_PORT0_P003	Connect OPAMP0 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP0_AMPM_TO_OPAMP0_AMPO	Connect OPAMP0 AMPM to OPAMP0 AMPO.
ANALOG_CONNECT_OPAMP0_AMPP_BREAK	Break all connections to OPAMP0 AMPP.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT5_P500	Connect OPAMP0 AMPP to PORT5 P500.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P014	Connect OPAMP0 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P013	Connect OPAMP0 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP0_AMPP_TO_PORT0_P002	Connect OPAMP0 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP0_AMPP_TO_DAC120_DA	Connect OPAMP0 AMPP to DAC120 DA.
ANALOG_CONNECT_OPAMP1_AMPM_BREAK	Break all connections to OPAMP1 AMPM.
ANALOG_CONNECT_OPAMP1_AMPM_TO_PORT0_P014	Connect OPAMP1 AMPM to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPM_TO_OPAMP1_AMPO	Connect OPAMP1 AMPM to OPAMP1 AMPO.
ANALOG_CONNECT_OPAMP1_AMPP_BREAK	Break all connections to OPAMP1 AMPP.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P014	Connect OPAMP1 AMPP to PORT0 P014.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P013	Connect OPAMP1 AMPP to PORT0 P013.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P003	Connect OPAMP1 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP1_AMPP_TO_PORT0_P002	Connect OPAMP1 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP1_AMPP_TO_DAC80_DA	Connect OPAMP1 AMPP to DAC80 DA.
ANALOG_CONNECT_OPAMP2_AMPM_BREAK	Break all connections to OPAMP2 AMPM.
ANALOG_CONNECT_OPAMP2_AMPM_TO_PORT0_P003	Connect OPAMP2 AMPM to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPM_TO_OPAMP2_AMPO	Connect OPAMP2 AMPM to OPAMP2 AMPO.

ANALOG_CONNECT_OPAMP2_AMPP_BREAK	Break all connections to OPAMP2 AMPP.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P003	Connect OPAMP2 AMPP to PORT0 P003.
ANALOG_CONNECT_OPAMP2_AMPP_TO_PORT0_P002	Connect OPAMP2 AMPP to PORT0 P002.
ANALOG_CONNECT_OPAMP2_AMPP_TO_DAC81_DA	Connect OPAMP2 AMPP to DAC81 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.

ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT1_P101	Connect ACMPLP0 IVREF0 to PORT1 P101.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_DAC80_DA	Connect ACMPLP0 IVREF0 to DAC80 DA.
ANALOG_CONNECT_ACMPLP0_IVREF0_TO_PORT5_P502	Connect ACMPLP0 IVREF0 to PORT5 P502.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT1_P103	Connect ACMPLP1 IVREF1 to PORT1 P103.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_DAC81_DA	Connect ACMPLP1 IVREF1 to DAC81 DA.
ANALOG_CONNECT_ACMPLP1_IVREF1_TO_PORT5_P500	Connect ACMPLP1 IVREF1 to PORT5 P500.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT1_P100	Connect ACMPLP0 IVCMP to PORT1 P100.
ANALOG_CONNECT_ACMPLP0_IVCMP_TO_PORT5_P503	Connect ACMPLP0 IVCMP to PORT5 P503.

ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP0 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT1_P102	Connect ACMPLP1 IVCMP to PORT1 P102.
ANALOG_CONNECT_ACMPLP1_IVCMP_TO_PORT5_P501	Connect ACMPLP1 IVCMP to PORT5 P501.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP0_IVREF0	Connect ACMPLP1 IVREF to ACMPLP0 IVREF0.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ACMPLP1_IVREF1	Connect ACMPLP1 IVREF to ACMPLP1 IVREF1.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P004	Connect ACMPHS0 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P007	Connect ACMPHS0 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P015	Connect ACMPHS0 IVCMP to PORT0 P015.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ANALOG0_VREF	Connect ACMPHS0 IVCMP to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P005	Connect ACMPHS0 IVREF to PORT0 P005.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P006	Connect ACMPHS0 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT0_P014	Connect ACMPHS0 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P000	Connect ACMPHS1 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P002	Connect ACMPHS1 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P003	Connect ACMPHS1 IVCMP to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P007	Connect ACMPHS1 IVCMP to PORT0 P007.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P015	Connect ACMPHS1 IVCMP to PORT0 P015.

ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P000	Connect ACMPHS1 IVREF to PORT0 P000.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P001	Connect ACMPHS1 IVREF to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P002	Connect ACMPHS1 IVREF to PORT0 P002.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P003	Connect ACMPHS1 IVREF to PORT0 P003.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P006	Connect ACMPHS1 IVREF to PORT0 P006.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT0_P014	Connect ACMPHS1 IVREF to PORT0 P014.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_ANALOG0_VREF	Connect ACMPLP0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP0_IVREF_TO_PORT1_P101	Connect ACMPLP0 IVREF to PORT1 P101.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_ANALOG0_VREF	Connect ACMPLP1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPLP1_IVREF_TO_PORT1_P103	Connect ACMPLP1 IVREF to PORT1 P103.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.

ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.

ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.

ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.

ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.

ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.

ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT5_P502	Connect ACMPHS0 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_DAC121_DA	Connect ACMPHS0 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_PORT0_P000	Connect ACMPHS0 IVCMP to PORT0 P000.
ANALOG_CONNECT_ACMPHS0_IVCMP_TO_ADC0_PGA0	Connect ACMPHS0 IVCMP to ADC0 PGA0.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P500	Connect ACMPHS0 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_PORT5_P501	Connect ACMPHS0 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_ANALOG0_VREF	Connect ACMPHS0 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS0_IVREF_TO_DAC120_DA	Connect ACMPHS0 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT5_P502	Connect ACMPHS1 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_DAC121_DA	Connect ACMPHS1 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_PORT0_P001	Connect ACMPHS1 IVCMP to PORT0 P001.
ANALOG_CONNECT_ACMPHS1_IVCMP_TO_ADC0_PGA1	Connect ACMPHS1 IVCMP to ADC0 PGA1.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P500	Connect ACMPHS1 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_PORT5_P501	Connect ACMPHS1 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_ANALOG0_VREF	Connect ACMPHS1 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS1_IVREF_TO_DAC120_DA	Connect ACMPHS1 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT5_P502	Connect ACMPHS2 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_DAC121_DA	Connect ACMPHS2 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_PORT0_P002	Connect ACMPHS2 IVCMP to PORT0 P002.
ANALOG_CONNECT_ACMPHS2_IVCMP_TO_ADC0_PGA2	Connect ACMPHS2 IVCMP to ADC0 PGA2.

ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P500	Connect ACMPHS2 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_PORT5_P501	Connect ACMPHS2 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_ANALOG0_VREF	Connect ACMPHS2 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS2_IVREF_TO_DAC120_DA	Connect ACMPHS2 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT5_P502	Connect ACMPHS3 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_DAC121_DA	Connect ACMPHS3 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_PORT0_P004	Connect ACMPHS3 IVCMP to PORT0 P004.
ANALOG_CONNECT_ACMPHS3_IVCMP_TO_ADC1_PGA3	Connect ACMPHS3 IVCMP to ADC1 PGA3.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P500	Connect ACMPHS3 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_PORT5_P501	Connect ACMPHS3 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_ANALOG0_VREF	Connect ACMPHS3 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS3_IVREF_TO_DAC120_DA	Connect ACMPHS3 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT5_P502	Connect ACMPHS4 IVCMP to PORT5 P502.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_DAC121_DA	Connect ACMPHS4 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_PORT0_P005	Connect ACMPHS4 IVCMP to PORT0 P005.
ANALOG_CONNECT_ACMPHS4_IVCMP_TO_ADC1_PGA4	Connect ACMPHS4 IVCMP to ADC1 PGA4.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P500	Connect ACMPHS4 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_PORT5_P501	Connect ACMPHS4 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_ANALOG0_VREF	Connect ACMPHS4 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS4_IVREF_TO_DAC120_DA	Connect ACMPHS4 IVREF to DAC120 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT5_P502	Connect ACMPHS5 IVCMP to PORT5 P502.

ANALOG_CONNECT_ACMPHS5_IVCMP_TO_DAC121_DA	Connect ACMPHS5 IVCMP to DAC121 DA.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_PORT0_P006	Connect ACMPHS5 IVCMP to PORT0 P006.
ANALOG_CONNECT_ACMPHS5_IVCMP_TO_ADC1_PGA5	Connect ACMPHS5 IVCMP to ADC1 PGA5.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P500	Connect ACMPHS5 IVREF to PORT5 P500.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_PORT5_P501	Connect ACMPHS5 IVREF to PORT5 P501.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_ANALOG0_VREF	Connect ACMPHS5 IVREF to ANALOG0 VREF.
ANALOG_CONNECT_ACMPHS5_IVREF_TO_DAC120_DA	Connect ACMPHS5 IVREF to DAC120 DA.

Cache Functions

Board Support Package » Supported MCUs » S7G2

Enumerations

enum bsp_cache_state_t

Detailed Description

This module implements cache functions.

Enumeration Type Documentation

◆ bsp_cache_state_t

enum bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

Clock Initialization

Board Support Package » Supported MCUs » S7G2

Macros

#define CGC_SRAM_ZERO_WAIT_CYCLES (0U)

Specify zero wait states for SRAM.

#define	CGC_SRAM_ONE_WAIT_CYCLES (1U) Specify one wait states for SRAM.
#define	CGC_ROM_ZERO_WAIT_CYCLES (0U) Specify zero wait states for ROM.
#define	CGC_ROM_ONE_WAIT_CYCLES (1U) Specify one wait states for ROM.
#define	CGC_ROM_TWO_WAIT_CYCLES (2U) Specify two wait states for ROM.

Functions

void	bsp_clock_init (void) Sets up system clocks. More...
uint32_t	bsp_cpu_clock_get (void) Returns frequency of CPU clock in Hz. More...
__STATIC_INLINE void	bsp_clocks_rom_wait_set (uint8_t setting) This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM. More...
__STATIC_INLINE uint32_t	bsp_clocks_rom_wait_get (void) This function gets the value of the ROMWT register. More...
__STATIC_INLINE void	bsp_clocks_sram_wait_set (uint32_t setting) This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory. More...
__STATIC_INLINE void	bsp_clocks_hsram_wait_set (uint32_t setting) This function sets the RAM wait state settings for High Speed RAM memory. More...

ssp_err_t	bsp_clock_set_callback (bsp_clock_set_callback_args_t *p_args) This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST) More...

Detailed Description

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Function Documentation

◆ bsp_clock_init()

void bsp_clock_init (void)

Sets up system clocks.

MOCO is default clock out of reset. Enable new clock if chosen.

Need to start PLL source clock and let it stabilize before starting PLL

Set PLL Divider.

Set PLL Multiplier.

Set PLL Source clock.

Wait for PLL clock source to stabilize

If the system clock has failed to start call the unrecoverable error handler.

MOCO, LOCO, and subclock do not have stabilization flags that can be checked.

Wait for clock source to stabilize

Set which clock to use for system clock and divisors for all system clocks.

If the system clock has failed to be configured properly call the unrecoverable error handler.

Set USB clock divisor.

Configure BCLK

Configure SDRAM Clock

◆ bsp_clock_set_callback()

ssp_err_t bsp_clock_set_callback (bsp_clock_set_callback_args_t * *p_args*)

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Parameters

[in]	p_args	Pre/Post request and clock frequency information

Return values

return	SSP_SUCCESS

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

< No MEMWAIT cycles

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

< No MEMWAIT cycles

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.

< No MEMWAIT cycles

Wait states for low speed RAM (SRAM0 and SRAM1)

No wait: ICLK <= 60 MHz

1 wait: ICLK > 60 MHz

Calculate the Wait states for ROM

Set the wait state BEFORE we change iclk

In this case we need to set the wait state AFTER we change iclk

Wait states for low speed RAM (SRAM0 and SRAM1)

No wait: ICLK <= 60 MHz

1 wait: ICLK > 60 MHz

Calculate the Wait states for ROM

Set the wait state BEFORE we change iclk

In this case we need to set the wait state AFTER we change iclk

Wait states for low speed RAM (SRAM0 and SRAM1)

No wait: ICLK <= 60 MHz

1 wait: ICLK > 60 MHz

Calculate the Wait states for ROM

Set the wait state BEFORE we change iclk

In this case we need to set the wait state AFTER we change iclk

◆ bsp_clocks_hsram_wait_set()

__STATIC_INLINE void bsp_clocks_hsram_wait_set (uint32_t *setting*)

This function sets the RAM wait state settings for High Speed RAM memory.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_clocks_rom_wait_get()

__STATIC_INLINE uint32_t bsp_clocks_rom_wait_get (void)

This function gets the value of the ROMWT register.

Return values

MEMWAIT	setting

◆ bsp_clocks_rom_wait_set()

__STATIC_INLINE void bsp_clocks_rom_wait_set (uint8_t *setting*)

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_clocks_sram_wait_set()

__STATIC_INLINE void bsp_clocks_sram_wait_set (uint32_t *setting*)

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Parameters

[in]	setting	The number of wait states to be used.

Return values

none	

◆ bsp_cpu_clock_get()

uint32_t bsp_cpu_clock_get (void)

Returns frequency of CPU clock in Hz.

Return values

Frequency	of the CPU in Hertz

Hardware Locks

Board Support Package » Supported MCUs » S7G2

Functions

SSP_HW_LOCK_DEFINE (ADC, 0U, 0U)

SSP_HW_LOCK_DEFINE (AGT, 0U, 0U)

SSP_HW_LOCK_DEFINE (BSC, 0U, 1U)

SSP_HW_LOCK_DEFINE (CAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CAN, 0U, 0U)

SSP_HW_LOCK_DEFINE (COMP_HS, 0U, 0U)

SSP_HW_LOCK_DEFINE (CRC, 0U, 0U)

SSP_HW_LOCK_DEFINE (CTSU, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAAD, 0U, 0U)

SSP_HW_LOCK_DEFINE (DAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DOC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DMAC, 0U, 0U)

SSP_HW_LOCK_DEFINE (DRW, 0U, 0U)

SSP_HW_LOCK_DEFINE (DTC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ELC, 0U, 0U)

SSP_HW_LOCK_DEFINE (EPTPC, 0U, 0U)

SSP_HW_LOCK_DEFINE (ETHER, 0U, 0U)

SSP_HW_LOCK_DEFINE (FCU, 0U, 0U)

SSP_HW_LOCK_DEFINE (GLCDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (GPT, 0U, 0U)

SSP_HW_LOCK_DEFINE (ICU, 0U, 0U)

SSP_HW_LOCK_DEFINE (IIC, 0U, 0U)

SSP_HW_LOCK_DEFINE (IRDA, 0U, 0U)

SSP_HW_LOCK_DEFINE (IWDT, 0U, 0U)

SSP_HW_LOCK_DEFINE (JPEG, 0U, 0U)

SSP_HW_LOCK_DEFINE (KEY, 0U, 0U)

SSP_HW_LOCK_DEFINE (LPM, 1U, 0U)

SSP_HW_LOCK_DEFINE (LVD, 0U, 0U)

SSP_HW_LOCK_DEFINE (MMF, 0U, 0U)

SSP_HW_LOCK_DEFINE (MPU, 0U, 0U)

SSP_HW_LOCK_DEFINE (OPS, 0U, 0U)

SSP_HW_LOCK_DEFINE (PDC, 0U, 0U)

SSP_HW_LOCK_DEFINE (POEG, 0U, 0U)
SSP_HW_LOCK_DEFINE (QSPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SPI, 0U, 0U)
SSP_HW_LOCK_DEFINE (RTC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCE, 0U, 0U)
SSP_HW_LOCK_DEFINE (SCI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SRC, 0U, 0U)
SSP_HW_LOCK_DEFINE (SSI, 0U, 0U)
SSP_HW_LOCK_DEFINE (SDHIMMC, 0U, 0U)
SSP_HW_LOCK_DEFINE (TSN, 0U, 0U)
SSP_HW_LOCK_DEFINE (USB, 0U, 0U)
SSP_HW_LOCK_DEFINE (WDT, 0U, 0U)

Detailed Description

This file allocates hardware locks used in Atomic Locking.

Function Documentation

◆ SSP_HW_LOCK_DEFINE() [1/44]

SSP_HW_LOCK_DEFINE (ADC , 0U , 0U)
Used to allocated hardware locks. Parameters are as follows: 1. IP name (ssp_ip_t enum without the SSP_IP_ prefix). 2. Unit number (used for blocks with variations like USB, not to be confused with ADC unit). 3. Channel numberADC

◆ SSP_HW_LOCK_DEFINE() [2/44]

SSP_HW_LOCK_DEFINE (AGT , 0U , 0U)
AGT

◆ SSP_HW_LOCK_DEFINE() [3/44]

SSP_HW_LOCK_DEFINE (BSC , 0U , 1U)
BSC

◆ SSP_HW_LOCK_DEFINE() [4/44]

SSP_HW_LOCK_DEFINE (CAC , 0U , 0U)
CAC

◆ SSP_HW_LOCK_DEFINE() [5/44]

SSP_HW_LOCK_DEFINE (CAN , 0U , 0U)
CAN

◆ SSP_HW_LOCK_DEFINE() [6/44]

SSP_HW_LOCK_DEFINE (COMP_HS , 0U , 0U)
COMP_HS

◆ SSP_HW_LOCK_DEFINE() [7/44]

SSP_HW_LOCK_DEFINE (CRC , 0U , 0U)
CRC

◆ SSP_HW_LOCK_DEFINE() [8/44]

SSP_HW_LOCK_DEFINE (CTSU , 0U , 0U)
CTSU

◆ SSP_HW_LOCK_DEFINE() [9/44]

SSP_HW_LOCK_DEFINE (DAAD , 0U , 0U)
DAAD

◆ SSP_HW_LOCK_DEFINE() [10/44]

SSP_HW_LOCK_DEFINE (DAC , 0U , 0U)
DAC

◆ SSP_HW_LOCK_DEFINE() [11/44]

SSP_HW_LOCK_DEFINE (DOC , 0U , 0U)
DOC

◆ SSP_HW_LOCK_DEFINE() [12/44]

SSP_HW_LOCK_DEFINE (DMAC , 0U , 0U)
DMAC

◆ SSP_HW_LOCK_DEFINE() [13/44]

SSP_HW_LOCK_DEFINE (DRW , 0U , 0U)
DRW

◆ SSP_HW_LOCK_DEFINE() [14/44]

SSP_HW_LOCK_DEFINE (DTC , 0U , 0U)
DTC

◆ SSP_HW_LOCK_DEFINE() [15/44]

SSP_HW_LOCK_DEFINE (ELC , 0U , 0U)
ELC

◆ SSP_HW_LOCK_DEFINE() [16/44]

SSP_HW_LOCK_DEFINE (EPTPC , 0U , 0U)
EPTPC

◆ SSP_HW_LOCK_DEFINE() [17/44]

SSP_HW_LOCK_DEFINE (ETHER , 0U , 0U)
ETHER

◆ SSP_HW_LOCK_DEFINE() [18/44]

SSP_HW_LOCK_DEFINE (FCU , 0U , 0U)
FCU

◆ SSP_HW_LOCK_DEFINE() [19/44]

SSP_HW_LOCK_DEFINE (GLCDC , 0U , 0U)
GLCDC

◆ SSP_HW_LOCK_DEFINE() [20/44]

SSP_HW_LOCK_DEFINE (GPT , 0U , 0U)
GPT

◆ SSP_HW_LOCK_DEFINE() [21/44]

SSP_HW_LOCK_DEFINE (ICU , 0U , 0U)
ICU

◆ SSP_HW_LOCK_DEFINE() [22/44]

SSP_HW_LOCK_DEFINE (IIC , 0U , 0U)
IIC

◆ SSP_HW_LOCK_DEFINE() [23/44]

SSP_HW_LOCK_DEFINE (IRDA , 0U , 0U)
IRDA

◆ SSP_HW_LOCK_DEFINE() [24/44]

SSP_HW_LOCK_DEFINE (IWDT , 0U , 0U)
IWDT

◆ SSP_HW_LOCK_DEFINE() [25/44]

SSP_HW_LOCK_DEFINE (JPEG , 0U , 0U)
JPEG

◆ SSP_HW_LOCK_DEFINE() [26/44]

SSP_HW_LOCK_DEFINE (KEY , 0U , 0U)
KEY

◆ SSP_HW_LOCK_DEFINE() [27/44]

SSP_HW_LOCK_DEFINE (LPM , 1U , 0U)
LPM

◆ SSP_HW_LOCK_DEFINE() [28/44]

SSP_HW_LOCK_DEFINE (LVD , 0U , 0U)
LVD

◆ SSP_HW_LOCK_DEFINE() [29/44]

SSP_HW_LOCK_DEFINE (MMF , 0U , 0U)
MMF

◆ SSP_HW_LOCK_DEFINE() [30/44]

SSP_HW_LOCK_DEFINE (MPU , 0U , 0U)
MPU

◆ SSP_HW_LOCK_DEFINE() [31/44]

SSP_HW_LOCK_DEFINE (OPS , 0U , 0U)
OPS

◆ SSP_HW_LOCK_DEFINE() [32/44]

SSP_HW_LOCK_DEFINE (PDC , 0U , 0U)
PDC

◆ SSP_HW_LOCK_DEFINE() [33/44]

SSP_HW_LOCK_DEFINE (POEG , 0U , 0U)
POEG

◆ SSP_HW_LOCK_DEFINE() [34/44]

SSP_HW_LOCK_DEFINE (QSPI , 0U , 0U)
QSPI

◆ SSP_HW_LOCK_DEFINE() [35/44]

SSP_HW_LOCK_DEFINE (SPI , 0U , 0U)
SPI

◆ SSP_HW_LOCK_DEFINE() [36/44]

SSP_HW_LOCK_DEFINE (RTC , 0U , 0U)
RTC

◆ SSP_HW_LOCK_DEFINE() [37/44]

SSP_HW_LOCK_DEFINE (SCE , 0U , 0U)
SCE

◆ SSP_HW_LOCK_DEFINE() [38/44]

SSP_HW_LOCK_DEFINE (SCI , 0U , 0U)
SCI

◆ SSP_HW_LOCK_DEFINE() [39/44]

SSP_HW_LOCK_DEFINE (SRC , 0U , 0U)
SRC

◆ SSP_HW_LOCK_DEFINE() [40/44]

SSP_HW_LOCK_DEFINE (SSI , 0U , 0U)
SSI

◆ SSP_HW_LOCK_DEFINE() [41/44]

SSP_HW_LOCK_DEFINE (SDHIMMC , 0U , 0U)
SDHIMMC

◆ SSP_HW_LOCK_DEFINE() [42/44]

SSP_HW_LOCK_DEFINE (TSN , 0U , 0U)
TSN

◆ SSP_HW_LOCK_DEFINE() [43/44]

SSP_HW_LOCK_DEFINE (USB , 0U , 0U)
USB

◆ SSP_HW_LOCK_DEFINE() [44/44]

SSP_HW_LOCK_DEFINE (WDT , 0U , 0U)
WDT

Module Start and Stop

Board Support Package » Supported MCUs » S7G2

Macros

#define	BSP_COMPILE_TIME_ASSERT(e) ((void) sizeof(char[1 - 2 * !(e)]))

Functions

ssp_err_t	R_BSP_ModuleStop (ssp_feature_t const *const p_feature) Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power. More...
ssp_err_t	R_BSP_ModuleStopAlways (ssp_feature_t const *const p_feature) Stop module (enter module stop) even if the module is used for multiple channels. More...
ssp_err_t	R_BSP_ModuleStart (ssp_feature_t const *const p_feature) Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set. More...
ssp_err_t	R_BSP_ModuleStateGet (ssp_feature_t const *const p_feature, bool *const p_stop)

Detailed Description

Module start and stop functions are provided to enable or disable peripherals.

Macro Definition Documentation

◆ BSP_COMPILE_TIME_ASSERT

#define BSP_COMPILE_TIME_ASSERT (e) ((void) sizeof(char[1 - 2 * !(e)]))
Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

Function Documentation

◆ R_BSP_ModuleStart()

ssp_err_t R_BSP_ModuleStart (ssp_feature_t const *const *p_feature*)

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

◆ R_BSP_ModuleStateGet()

ssp_err_t R_BSP_ModuleStateGet (ssp_feature_t const *const *p_feature*, bool *const *p_stop*)

The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.

Save the current module state

◆ R_BSP_ModuleStop()

ssp_err_t R_BSP_ModuleStop (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Note

Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

◆ R_BSP_ModuleStopAlways()

ssp_err_t R_BSP_ModuleStopAlways (ssp_feature_t const *const *p_feature*)

Stop module (enter module stop) even if the module is used for multiple channels.

Parameters

[in]	p_feature	Pointer to definition of the feature, defined by ssp_feature_t.

Return values

SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid

ROM Registers

Board Support Package » Supported MCUs » S7G2

Macros

#define	BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))

Detailed Description

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Macro Definition Documentation

◆ BSP_ROM_REG_OFS1_SETTING

#define BSP_ROM_REG_OFS1_SETTING (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFFF9FFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 9))
OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

5.2.2 Common BSP Code

Board Support Package

Code common to all BSPs. More...

Modules

	Common BSP LED Code and Types Common support for board LEDs.
	Compiler Support
	Software Delay Common function to implement a software delay.
	Error Checking
	Module specific feature overrides
	Grouped Interrupt Support
	Interrupt Initialization
	Atomic Locking

	Register Protection
	BSP_MCU_SBRK

Macros

#define	BSP_IRQ_DISABLED (0xFFU)
#define	SSP_ASSERT_FAIL
#define	SSP_ERROR_LOG(err, module, version)
#define	SSP_ASSERT(a)
#define	SSP_ERROR_RETURN(a, err, module, version)
#define	SSP_CRITICAL_SECTION_DEFINE uint32_t old_mask_level = 0U
#define	SSP_CRITICAL_SECTION_ENTER
#define	SSP_CRITICAL_SECTION_EXIT __set_PRIMASK(old_mask_level)

Functions

ssp_err_t	R_BSP_VersionGet (ssp_version_t *p_version) Set BSP version based on compile time macros. More...
ssp_err_t	R_SSP_VersionGet (ssp_pack_version_t *const p_version) Set SSP version based on compile time macros. More...
void	bsp_init_internal (void *p_args) Default initialization function, used only if bsp_init is not defined in the user application.

Detailed Description

Code common to all BSPs.

Implements functions that are common to all BSPs.

Macro Definition Documentation

◆ BSP_IRQ_DISABLED

#define BSP_IRQ_DISABLED (0xFFU)
Used to signify that an ELC event is not able to be used as an interrupt.

◆ SSP_ASSERT

#define SSP_ASSERT (a)

```
{ \
      if ((a))                      \
      {                            \
           (void) 0;/* Do nothing */ \
      }                            \
      else                         \
      {                            \
           SSP_ASSERT_FAIL;          \
           return SSP_ERR_ASSERTION; \
      }                            \
   }
```

Default assertion calls SSP_ASSERT_FAIL if condition "a" is false. Used to identify incorrect use of API's in SSP functions.

◆ SSP_ASSERT_FAIL

#define SSP_ASSERT_FAIL
Function call to insert before returning assertion error.

◆ SSP_CRITICAL_SECTION_DEFINE

#define SSP_CRITICAL_SECTION_DEFINE uint32_t old_mask_level = 0U
This check is performed to select suitable ASM API with respect to core This macro defines a variable for saving previous mask value

◆ SSP_CRITICAL_SECTION_ENTER

#define SSP_CRITICAL_SECTION_ENTER

```
old_mask_level = __get_PRIMASK(); \
      __set_PRIMASK(1U)
```

This macro defined to get the mask value

◆ SSP_CRITICAL_SECTION_EXIT

#define SSP_CRITICAL_SECTION_EXIT __set_PRIMASK(old_mask_level)

This macro defined to restore the old mask value

◆ SSP_ERROR_LOG

#define SSP_ERROR_LOG (err, module, version)

This function is called before returning an error code. To stop on a runtime error, define ssp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function.

◆ SSP_ERROR_RETURN

#define SSP_ERROR_RETURN (a, err, module, version)

```
{ \
      if ((a))                                  \
      {                                        \
           (void) 0; /* Do nothing */           \
     } \
      else                                     \
      {                                        \
           SSP_ERROR_LOG((err), (module), (version)); \
           return (err);                         \
     } \
   }
```

All SSP error codes are returned using this macro. Calls SSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in SSP functions.

Function Documentation

◆ R_BSP_VersionGet()

ssp_err_t R_BSP_VersionGet (ssp_version_t * *p_version*)

Set BSP version based on compile time macros.

Parameters

[out]	p_version	Memory address to return version information to.

Return values

SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ R_SSP_VersionGet()

ssp_err_t R_SSP_VersionGet (ssp_pack_version_t *const *p_version*)

Set SSP version based on compile time macros.

Parameters

[out]	p_version	Memory address to return version information to.

Return values

SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

5.2.2.1 Common BSP LED Code and Types

Board Support Package » Common BSP Code

Common support for board LEDs. More...

Data Structures

struct	bsp_leds_t

ssp_err_t	R_BSP_LedsGet (bsp_leds_t *p_leds)

	Return information about the LEDs on the current board. More...

Detailed Description

Common support for board LEDs.

Contains types and functions that allow for common use of LEDs on boards

Function Documentation

◆ R_BSP_LedsGet()

ssp_err_t R_BSP_LedsGet (bsp_leds_t * *p_leds*)

Return information about the LEDs on the current board.

Structure with LED information.

Parameters

[out]	p_leds	Pointer to structure where LED info is stored.

bsp_leds_t Struct Reference

Board Support Package » Common BSP Code » Common BSP LED Code and Types

#include <bsp_common_leds.h>

Data Fields

uint16_t	led_count The number of LEDs on this board.
ioport_port_pin_t const *	p_leds Pointer to an array of IOPORT pins for controlling LEDs.

Detailed Description

Information on how many LEDs and what pins they are on.

The documentation for this struct was generated from the following file:

- bsp_common_leds.h

5.2.2.2 Compiler Support

Board Support Package » Common BSP Code

Detailed Description

The macros in this file are defined based upon which compiler is being used. The macros abstract common section names and gives a common way to place code in a particular section. Some macros have a version that ends in V2. These were created to unify the usage between compilers while retaining backwards compatibility in the existing macros and should be preferred in new code.

Description of macros:

- BSP_SECTION_STACK - Name of section where stack(s) are stored
- BSP_SECTION_HEAP - Name of section where heap(s) are stored
- BSP_SECTION_VECTOR - Name of section where vector table is stored
- BSP_SECTION_ROM_REGISTERS - Name of section where ROM registers are located
- BSP_PLACE_IN_SECTION - Macro for placing code in a particular section
- BSP_ALIGN_VARIABLE - Macro for specifiying a minimum alignment in bytes
- BSP_PACKED - Macro for setting a 1 byte alignment to remove padding
- BSP_DONT_REMOVE - Keyword to tell linker/compiler to not optimize out a variable or function

Note

Currently supported compilers are GCC and IAR

5.2.2.3 Software Delay

Board Support Package » Common BSP Code

Common function to implement a software delay. More...

Functions

void	R_BSP_SoftwareDelay (uint32_t delay, bsp_delay_units_t units) Delay the specified duration in units and return. More...

Detailed Description

Common function to implement a software delay.

Implements a software delay function for all BSPs.

Function Documentation

◆ R_BSP_SoftwareDelay()

void R_BSP_SoftwareDelay (uint32_t *delay*, bsp_delay_units_t *units*)

Delay the specified duration in units and return.

Parameters

[in]	delay	The number of 'units' to delay.
[in]	units	The 'base' (bsp_delay_units_t) for the units specified. Valid values are: BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS. For example: At 1 MHz one cycle takes 1 microsecond (.000001 seconds). At 12 MHz one cycle takes 1/12 microsecond or 83 nanoseconds. Therefore one run through software_delay_loop() takes: ~ (83 * DELAY_LOOP_CYCLES) or 332 ns. A delay of 2 us therefore requires 2000ns/332ns or 6 loops.

The 'theoretical' maximum delay that may be obtained is determined by a full 32 bit loop count and the system clock rate. @240MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 240000000) = 71 seconds. @32MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 32000000) = 536 seconds

Note that requests for very large delays will be affected by rounding in the calculations and the actual delay achieved may be slightly less. @32 MHz, for example, a request for 532 seconds will be closer to 536 seconds.

Note also that if the calculations result in a loop_cnt of zero, the software_delay_loop() function is not called at all. In this case the requested delay is too small (nanoseconds) to be carried out by the loop itself, and the overhead associated with executing the code to just get to this point has certainly satisfied the requested delay.

Note

This function calls bsp_cpu_clock_get() which ultimately calls R_CGC_SystemClockFreqGet() and therefore requires that the BSP has already initialized the CGC (which it does as part of the Sysinit). Care should be taken to ensure this remains the case if in the future this function were to be called as part of the BSP initialization.

Return values

None.	

Convert the requested time to microseconds.

Get the system clock frequency in Hz.

Get the # of nanoseconds/cycle.

Only delay if the supplied parameters constitute a delay.

5.2.2.4 Error Checking

Board Support Package » Common BSP Code

Macros

#define	BSP_STACK_ALIGNMENT (8)

Detailed Description

This file performs build time error checking where possible.

Macro Definition Documentation

◆ BSP_STACK_ALIGNMENT

#define BSP_STACK_ALIGNMENT (8)
Stacks (and heap) must be sized and aligned to an integer multiple of this number.

5.2.2.5 Module specific feature overrides

Board Support Package » Common BSP Code

Data Structures

struct	bsp_feature_sci_t
struct	bsp_feature_rspi_t
struct	bsp_feature_lvd_t
struct	bsp_feature_acmphs_t
struct	bsp_feature_adc_t
struct	bsp_feature_can_t

struct	bsp_feature_dac_t
struct	bsp_feature_flash_lp
struct	bsp_feature_flash_hp
struct	bsp_feature_ctsu_t
struct	bsp_feature_ioport_t
struct	bsp_feature_cgc_t
struct	bsp_feature_opamp_t
struct	bsp_feature_sdhi_t
struct	bsp_feature_ssi_t
struct	bsp_feature_icu_t
struct	bsp_feature_lpmv2_t
struct	bsp_feature_riic_t

Functions

void	R_BSP_FeatureSciGet (bsp_feature_sci_t *p_sci_feature)
void	R_BSP_FeatureRspiGet (bsp_feature_rspi_t *p_rspi_feature)
void	R_BSP_FeatureLvdGet (bsp_feature_lvd_t *p_lvd_feature)
void	R_BSP_FeatureAcmphsGet (bsp_feature_acmphs_t *p_acmphs_feature)
void	R_BSP_FeatureAdcGet (bsp_feature_adc_t *p_adc_feature)
void	R_BSP_FeatureCtsuGet (bsp_feature_ctsu_t *p_ctsu_feature)
void	R_BSP_FeatureIoportGet (bsp_feature_ioport_t *p_ioport_feature)
void	R_BSP_FeatureCgcGet (bsp_feature_cgc_t const **pp_cgc_feature)
void	R_BSP_FeatureCanGet (bsp_feature_can_t *p_can_feature)
void	R_BSP_FeatureDacGet (bsp_feature_dac_t *p_dac_feature)
void	R_BSP_FeatureFlashLpGet (bsp_feature_flash_lp *p_flash_lp_feature)

void	R_BSP_FeatureOpampGet (bsp_feature_opamp_t *const p_opamp_feature)
void	R_BSP_FeatureSdhiGet (bsp_feature_sdhi_t *p_sdhi_feature)
void	R_BSP_FeatureSsiGet (bsp_feature_ssi_t *p_ssi_feature)
void	R_BSP_FeatureICUGet (bsp_feature_icu_t *p_icu_feature)
void	R_BSP_FeatureLPMV2Get (bsp_feature_lpmv2_t *p_lpmv2_feature)
void	R_BSP_FeatureRIICGet (bsp_feature_riic_t *p_riic_feature)

Detailed Description

This group contains lookup functions that provide MCU specific feature information that is not available in the factory flash.

Function Documentation

◆ R_BSP_FeatureAcmphsGet()

void R_BSP_FeatureAcmphsGet (bsp_feature_acmphs_t * *p_acmphs_feature*)

Get MCU specific ACMPHS features

Parameters

[out]	p_acmphs_feature	Pointer to structure to store ACMPHS features.

◆ R_BSP_FeatureAdcGet()

void R_BSP_FeatureAdcGet (bsp_feature_adc_t * *p_adc_feature*)

Get MCU specific ADC features

Parameters

[out]	p_adc_feature	Pointer to structure to store ADC features.

◆ R_BSP_FeatureCanGet()

void R_BSP_FeatureCanGet (bsp_feature_can_t * *p_can_feature*)

Get MCU specific CAN features

Parameters

[out]	p_can_feature	Pointer to structure to store CAN features.

◆ R_BSP_FeatureCgcGet()

void R_BSP_FeatureCgcGet (bsp_feature_cgc_t const ** *pp_cgc_feature*)

Get MCU specific CGC features

Parameters

[out]	pp_cgc_feature	Pointer to structure to store pointer to CGC features.

◆ R_BSP_FeatureCtsuGet()

void R_BSP_FeatureCtsuGet (bsp_feature_ctsu_t * *p_ctsu_feature*)

Get MCU specific CTSU features

Parameters

[out]	p_ctsu_feature	Pointer to structure to store CTSU features.

◆ R_BSP_FeatureDacGet()

void R_BSP_FeatureDacGet (bsp_feature_dac_t * *p_dac_feature*)

Get MCU specific DAC features

Parameters

[out]	p_dac_feature	Pointer to structure to store DAC features.

◆ R_BSP_FeatureFlashLpGet()

void R_BSP_FeatureFlashLpGet (bsp_feature_flash_lp * *p_flash_lp_feature*)

Get MCU specific FLASH LP features

Parameters

[out]	p_flash_lp_feature	Pointer to structure to store Flash LP features.

S124 uses 1 macro of 128K and single access for Code Flash. It can therefore access 128K as a single macro and it's Code Flash memory is effectively organized as a single macro of 128K, yielding a total of 128K Code Flash.

S128 uses 1 macro of 256K and single access for Code Flash. It can therefore access 256K as a single macro and it's Code Flash memory is effectively organized as a single macro of 256K, yielding a total of 256K Code Flash.

S1JA uses 2 macros of 128K and double access for Code Flash. It can therefore access 256K as a single macro and it's Code Flash memory is effectively organized as 1 macro of 256K each, yielding a total of 256K Code Flash.

S3A1 uses 4 macros of 256K and double access for Code Flash. It can therefore access 512K as a single macro and it's Code Flash memory is effectively organized as 2 macros of 512K each, yielding a total of 1MB Code Flash. This generates a macro boundary at 512K which is important for blank checking.

S3A7 uses 4 macros of 256K and double access for Code Flash. It can therefore access 512K as a single macro and it's Code Flash memory is effectively organized as 2 macros of 512K each, yielding a total of 1MB Code Flash. This generates a macro boundary at 512K which is important for blank checking.

◆ R_BSP_FeatureICUGet()

void R_BSP_FeatureICUGet (bsp_feature_icu_t * *p_icu_feature*)

Get MCU specific DMAC features

Parameters

[out]	p_icu_feature	Pointer to structure to store DMAC features.

◆ R_BSP_FeatureIoportGet()

void R_BSP_FeatureIoportGet (bsp_feature_ioport_t * *p_ioport_feature*)

Get MCU specific I/O port features

Parameters

[out]	p_ioport_feature	Pointer to structure to store I/O port features.

◆ R_BSP_FeatureLPMV2Get()

void R_BSP_FeatureLPMV2Get (bsp_feature_lpmv2_t * *p_lpmv2_feature*)

Get MCU specific LPMV2 features

Parameters

[out]	p_lpmv2_feature	Pointer to structure to store LPMV2 features.

◆ R_BSP_FeatureLvdGet()

void R_BSP_FeatureLvdGet (bsp_feature_lvd_t * *p_lvd_feature*)

Get MCU specific LVD features.

Parameters

[out]	p_lvd_feature	Pointer to structure to store LVD features.

◆ R_BSP_FeatureOpampGet()

void R_BSP_FeatureOpampGet (bsp_feature_opamp_t *const *p_opamp_feature*)

Get MCU specific OPAMP features

Parameters

[out]	p_opamp_feature	Pointer to structure to store OPAMP features.

◆ R_BSP_FeatureRIICGet()

void R_BSP_FeatureRIICGet (bsp_feature_riic_t * *p_riic_feature*)

Get MCU specific RIIC features

Parameters

[out]	p_riic_feature	Pointer to structure to store RIIC features.

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

< Initialize the input rise time for standard and fast mode

< Initialize the input rise time for fastplus mode

◆ R_BSP_FeatureRspiGet()

void R_BSP_FeatureRspiGet (bsp_feature_rspi_t * *p_rspi_feature*)

Get MCU specific RSPI features.

Parameters

[out]	p_rspi_feature	Pointer to structure to store RSPI features.

◆ R_BSP_FeatureSciGet()

void R_BSP_FeatureSciGet (bsp_feature_sci_t * *p_sci_feature*)

Get MCU specific SCI features.

Parameters

[out]	p_sci_feature	Pointer to structure to store SCI features.

◆ R_BSP_FeatureSdhiGet()

void R_BSP_FeatureSdhiGet (bsp_feature_sdhi_t * *p_sdhi_feature*)

Get MCU specific SDHI features

Parameters

[out]	p_sdhi_feature	Pointer to structure to store SDHI features.

◆ R_BSP_FeatureSsiGet()

void R_BSP_FeatureSsiGet (bsp_feature_ssi_t * *p_ssi_feature*)

Get MCU specific SSI features

Parameters

[out]	p_ssi_feature	Pointer to structure to store SSI features.

bsp_feature_sci_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	clock Which clock the SCI is connected to.

Detailed Description

SCI MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_rspi_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	clock Which clock the RSPI is connected to.

Detailed Description

RSPI MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_lvd_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	monitor_1_low_threshold Monitor 1 lowest valid voltage threshold.
uint8_t	monitor_1_hi_threshold Monitor 1 highest valid voltage threshold.
uint8_t	monitor_2_low_threshold Monitor 2 lowest valid voltage threshold.
uint8_t	monitor_2_hi_threshold Monitor 2 highest valid voltage threshold.
uint32_t	has_digital_filter: 1 Whether or not LVD has a digital filter.
uint32_t	negation_delay_clock Clock required for LVD signal negation delay after reset.

Detailed Description

LVD MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_acmphs_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint32_t	min_wait_time_us Minimum stabilization wait time in microseconds.

Detailed Description

ACMPHS MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_adc_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	has_sample_hold_reg: 1 Whether or not sample and hold registers are present.
uint8_t	group_b_sensors_allowed: 1 Whether or not sensors are allowed on group b.
uint8_t	sensors_exclusive: 1 Whether or not sensors can be used with other sensors/channels.
uint8_t	tsn_calibration_available: 1 Identify if the TSN calibration data is available.
uint8_t	tsn_control_available: 1 Identify if the TSN control register is available.

int16_t	tsn_slope TSN slope in micro-volts/°C.
uint32_t	sensor_min_sampling_time The minimum sampling time required by the on-chip temperature and voltage sensor in nsec.
uint32_t	clock_source The conversion clock used by the ADC peripheral.
uint8_t	addition_supported: 1 Whether addition is supported or not in this MCU.
uint8_t	calibration_reg_available: 1 Whether CALEXE register is available.
uint8_t	reference_voltage: 1 Whether external or internal ref voltage.

Detailed Description

ADC MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_can_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	mclock_only: 1

	Whether or not MCLK is the only valid clock.
uint8_t	check_pclkb_ratio: 1 Whether clock:PCLKB must be 2:1.
uint8_t	clock Which clock to compare PCLKB to.

Detailed Description

CAN MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_dac_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	has_davrefcr: 1 Whether or not DAC has DAVREFCR register.
uint8_t	has_chargepump: 1 Whether or not DAC has DAPC register.

Detailed Description

DAC MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_flash_lp Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	flash_clock_src Source clock for Flash (ie. FCLK or ICLK)
uint8_t	flash_cf_macros Number of implemented code flash hardware macros.
uint32_t	cf_macro_size The size of the implemented Code Flash macro.

Detailed Description

FLASH LP MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_flash_hp Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	cf_block_size_write Code Flash Block size write.

Detailed Description

FLASH HP MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_ctsu_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	ctsucr0_mask Mask of valid bits in CTSUCR0.
uint8_t	ctsucr1_mask Mask of valid bits in CTSUCR1.
uint8_t	ctsumch0_mask Mask of valid bits in CTSUMCH0.
uint8_t	ctsumch1_mask Mask of valid bits in CTSUMCH1.
uint8_t	ctsuchac_register_count Number of CTSUCHAC registers.
uint8_t	ctsuchtrc_register_count Number of CTSUCHTRC registers.

Detailed Description

CTSU MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_ioport_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	has_ethernet: 1 Whether or not MCU has Ethernet port configurations.
uint8_t	has_vbatt_pins: 1 Whether or not MCU has pins on vbatt domain.

Detailed Description

I/O port MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_cgc_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint32_t	high_speed_freq_hz Frequency above which high speed mode must be used.

uint32_t	middle_speed_max_freq_hz Max frequency for middle speed, 0 indicates not available.
uint32_t	low_speed_max_freq_hz Max frequency for low speed, 0 indicates not available.
uint32_t	low_voltage_max_freq_hz Max frequency for low voltage, 0 indicates not available.
uint32_t	low_speed_pclk_div_min Minimum divisor for peripheral clocks when using oscillator stop detect.
uint32_t	low_voltage_pclk_div_min Minimum divisor for peripheral clocks when using oscillator stop detect.
uint32_t	hoco_freq_hz HOCO frequency.
uint32_t	main_osc_freq_hz Main oscillator frequency.
uint8_t	modrv_mask Mask for MODRV in MOMCR.
uint8_t	modrv_shift Offset of lowest bit of MODRV in MOMCR.
uint8_t	sodrv_mask Mask for SODRV in SOMCR.
uint8_t	sodrv_shift Offset of lowest bit of SODRV in SOMCR.

uint8_t	pll_div_max Maximum PLL divisor.
uint8_t	pll_mul_min Minimum PLL multiplier.
uint8_t	pll_mul_max Maximum PLL multiplier.
uint8_t	mainclock_drive Main clock drive capacity.
uint32_t	iclk_div: 4 ICLK divisor.
uint32_t	pllccr_type: 2 0: No PLL, 1: PLLCCR, 2: PLLCCR2
uint32_t	pll_src_configurable: 1 Whether or not PLL clock source is configurable.
uint32_t	has_subosc_speed: 1 Whether or not MCU has subosc speed mode.
uint32_t	has_lcd_clock: 1 Whether or not MCU has LCD clock.
uint32_t	has_sdram_clock: 1 Whether or not MCU has SDRAM clock.
uint32_t	has_usb_clock_div: 1 Whether or not MCU has USB clock divisor.

uint32_t	has_pclka: 1 Whether or not MCU has PCLKA clock.
uint32_t	has_pclkb: 1 Whether or not MCU has PCLKB clock.
uint32_t	has_pclkc: 1 Whether or not MCU has PCLKC clock.
uint32_t	has_pclkd: 1 Whether or not MCU has PCLKD clock.
uint32_t	has_fclk: 1 Whether or not MCU has FCLK clock.
uint32_t	has_bclk: 1 Whether or not MCU has BCLK clock.
uint32_t	has_sdadc_clock: 1 Whether or not MCU has SDADC clock.
uint32_t	set_bck_with_pckb: 1 Whether or not BCK bits should be set with PCKB bits.

Detailed Description

CGC MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_opamp_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint16_t	min_wait_time_lp_us Minimum wait time in low power mode.
uint16_t	min_wait_time_ms_us Minimum wait time in middle speed mode.
uint16_t	min_wait_time_hs_us Minimum wait time in high speed mode.

Detailed Description

OPAMP MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_sdhi_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint32_t	has_card_detection: 1 Whether or not MCU has card detection.
uint32_t	supports_8_bit_mmc: 1 Whether or not MCU supports 8-bit MMC.
uint32_t	max_clock_frequency

	Maximum clock rate supported by the peripheral.

Detailed Description

SDHI MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_ssi_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	fifo_num_stages Number of FIFO stages on this MCU.

Detailed Description

SSI MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_icu_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	has_ir_flag: 1

	Whether or not MCU has IR flag in DELSRn register.

Detailed Description

DMAC MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_lpmv2_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint8_t	has_dssby: 1 Whether or not MCU has Deep software standby mode.

Detailed Description

LPMV2 MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

bsp_feature_riic_t Struct Reference

Board Support Package » Common BSP Code » Module specific feature overrides

#include <bsp_feature.h>

Data Fields

uint32_t	riic_std_fast_rise_time

	Input rise time for the riic peripheral for standard and fast mode.
uint32_t	riic_fastplus_rise_time Input rise time for the riic peripheral for the fast plus mode.

Detailed Description

RIIC MCU specific features.

The documentation for this struct was generated from the following file:

- bsp_feature.h

5.2.2.6 Grouped Interrupt Support

Board Support Package » Common BSP Code

Functions

ssp_err_t	R_BSP_GroupIrqWrite (bsp_grp_irq_t irq, void(*p_callback)(bsp_grp_irq_t irq)) Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered. More...
void	NMI_Handler (void) Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception. More...

Detailed Description

Support for grouped interrupts. Grouped interrupts occur when multiple interrupt events trigger the same interrupt vector. When this common vector is triggered the activation source must be discovered. The functions in this file allow users to register a callback function for a single interrupt source in an interrupt group.

Function Documentation

◆ NMI_Handler()

void NMI_Handler (void)
Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception. Determine what is the cause of this interrupt. IWDT underflow/refresh error interrupt is requested. Clear IWDT flag. WDT underflow/refresh error interrupt is requested. Clear WDT flag. Voltage monitoring 1 interrupt is requested. Clear LVD1 flag. Voltage monitoring 2 interrupt is requested. Clear LVD2 flag. Oscillation stop detection interrupt is requested. Clear oscillation stop detect flag. NMI pin interrupt is requested. Clear NMI pin interrupt flag. RAM Parity Error interrupt is requested. Clear RAM parity error flag. RAM ECC Error interrupt is requested. Clear RAM ECC error flag. Determine what is the cause of this interrupt. IWDT underflow/refresh error interrupt is requested. Clear IWDT flag. WDT underflow/refresh error interrupt is requested. Clear WDT flag. Voltage monitoring 1 interrupt is requested. Clear LVD1 flag. Voltage monitoring 2 interrupt is requested. Clear LVD2 flag. Oscillation stop detection interrupt is requested. Clear oscillation stop detect flag. NMI pin interrupt is requested. Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

Determine what is the cause of this interrupt.

IWDT underflow/refresh error interrupt is requested.

Clear IWDT flag.

WDT underflow/refresh error interrupt is requested.

Clear WDT flag.

Voltage monitoring 1 interrupt is requested.

Clear LVD1 flag.

Voltage monitoring 2 interrupt is requested.

Clear LVD2 flag.

VBATT Monitor interrupt is requested.

Clear VBATT flag.

Oscillation stop detection interrupt is requested.

Clear oscillation stop detect flag.

NMI pin interrupt is requested.

Clear NMI pin interrupt flag.

RAM Parity Error interrupt is requested.

Clear RAM parity error flag.

RAM ECC Error interrupt is requested.

Clear RAM ECC error flag.

MPU Bus Slave Error interrupt is requested.

Clear MPU Bus Slave error flag.

MPU Bus Master Error interrupt is requested.

Clear MPU Bus Master error flag.

MPU Stack Error interrupt is requested.

Clear MPU Stack error flag.

◆ R_BSP_GroupIrqWrite()

ssp_err_t R_BSP_GroupIrqWrite (bsp_grp_irq_t *irq*, void(*)(bsp_grp_irq_t irq) *p_callback*)

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

Parameters

[in]	irq	Interrupt for which to register a callback.
[in]	p_callback	Pointer to function to call when interrupt occurs.

Return values

SSP_SUCCESS	Callback registered

Check for valid address.

Callback was NULL. De-register callback.

Register callback.

Check for valid address.

Callback was NULL. De-register callback.

Register callback.

Check for valid address.

Callback was NULL. De-register callback.

Register callback.

5.2.2.7 Interrupt Initialization

Board Support Package » Common BSP Code

Functions

void	R_BSP_IrqStatusClear (IRQn_Type irq) Clear the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately. More...
void	bsp_irq_cfg (void) Using the vector table information section that has been built by the linker and placed into ROM in the .vector_info. section, this function will initialize the ICU so that configured ELC events will trigger

	interrupts in the NVIC.

Detailed Description

This module configures certain ELC events so that they can trigger NVIC interrupts. Which events are used as interrupts depends on settings in bsp_irq_cfg.h.

Function Documentation

◆ R_BSP_IrqStatusClear()

void R_BSP_IrqStatusClear (IRQn_Type *irq*)

Clear the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.

Parameters

[in]	irq	Interrupt for which to clear the IR bit. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. In prior releases this list contained all of the interrupts enabled for the specfic MCU but enabled interrupts are now configured using the SSP_VECTOR_DEFINE macro.

Note

This does not work for system exceptions where the IRQn_Type value is < 0.

5.2.2.8 Atomic Locking

Board Support Package » Common BSP Code

Data Structures

struct	bsp_lock_t

Functions

ssp_err_t	R_BSP_SoftwareLock (bsp_lock_t *p_lock) Attempt to acquire the lock that has been sent in. More...
void	R_BSP_SoftwareUnlock (bsp_lock_t *p_lock)

	Release hold on lock. More...
ssp_err_t	R_BSP_HardwareLock (ssp_feature_t const *const p_feature) Attempt to reserve a hardware resource lock. More...
void	R_BSP_HardwareUnlock (ssp_feature_t const *const p_feature) Release hold on lock. More...
void	bsp_init_hardware_locks (void) Initialize all of the hardware locks to BSP_LOCK_UNLOCKED.
void	R_BSP_SoftwareLockInit (bsp_lock_t *p_lock) Initialize lock value to be unlocked. More...

Detailed Description

This module implements atomic locking mechanisms.

Function Documentation

◆ R_BSP_HardwareLock()

ssp_err_t R_BSP_HardwareLock (ssp_feature_t const *const *p_feature*)

Attempt to reserve a hardware resource lock.

Parameters

[in]	p_feature	Pointer to the module specific feature information.

Return values

SSP_SUCCESS	Lock was acquired
SSP_ERR_IN_USE	Lock was not acquired

◆ R_BSP_HardwareUnlock()

void R_BSP_HardwareUnlock (ssp_feature_t const *const *p_feature*)

Release hold on lock.

Parameters

[in]	p_feature	Pointer to the module specific feature information.

◆ R_BSP_SoftwareLock()

ssp_err_t R_BSP_SoftwareLock (bsp_lock_t * *p_lock*)

Attempt to acquire the lock that has been sent in.

Parameters

[in]	p_lock	Pointer to the structure which contains the lock to be acquired.

Return values

SSP_SUCCESS	Lock was acquired
SSP_ERR_IN_USE	Lock was not acquired

◆ R_BSP_SoftwareLockInit()

void R_BSP_SoftwareLockInit (bsp_lock_t * *p_lock*)

Initialize lock value to be unlocked.

Parameters

[in]	p_lock	Pointer to the structure which contains the lock to initialize.

◆ R_BSP_SoftwareUnlock()

void R_BSP_SoftwareUnlock (bsp_lock_t * *p_lock*)
Release hold on lock. **Parameters** [in] p_lock — Pointer to the structure which contains the lock to unlock.

bsp_lock_t Struct Reference

Board Support Package » Common BSP Code » Atomic Locking

#include <bsp_locking.h>

Data Fields

uint8_t	lock A uint8_t is used instead of a enum because the size must be 8-bits.

Detailed Description

Lock structure. Passed into software locking functions such as R_BSP_SoftwareLock() and R_BSP_SoftwareUnLock.

The documentation for this struct was generated from the following file:

- bsp_locking.h

5.2.2.9 Register Protection

Board Support Package » Common BSP Code

Functions

void	R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect) Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). More...

void	R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect) Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). More...
void	bsp_register_protect_open (void) Initializes variables needed for register protection functionality. More...

Detailed Description

Important registers are write protected. This module provides APIs for configuring the protection of these registers. Reference counters are used to ensure proper operation.

Function Documentation

◆ bsp_register_protect_open()

void bsp_register_protect_open (void)
Initializes variables needed for register protection functionality. Initialize reference counters to 0.

◆ R_BSP_RegisterProtectDisable()

void R_BSP_RegisterProtectDisable (bsp_reg_protect_t *regs_to_unprotect*)
Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). **Parameters** [in] regs_to_unprotect — Registers which have write protection disabled. Get/save the current state of interrupts Disable protection using PRCR register. When writing to the PRCR register the upper 8-bits must be the correct key. Set lower bits to 0 to disable writes. Increment the protect counter Restore the interrupt state

◆ R_BSP_RegisterProtectEnable()

void R_BSP_RegisterProtectEnable (bsp_reg_protect_t *regs_to_protect*)
Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). **Parameters** [in] regs_to_protect — Registers which have write protection enabled. Get/save the current state of interrupts Enable protection using PRCR register. When writing to the PRCR register the upper 8-bits must be the correct key. Set lower bits to 0 to disable writes. Restore the interrupt state

5.2.2.10 BSP_MCU_SBRK

Board Support Package » Common BSP Code

Functions

caddr_t _sbrk (int incr)

SSP implementation of the standard library _sbrk() function. More...

Detailed Description

Function Documentation

◆ _sbrk()

caddr_t _sbrk (int *incr*)

SSP implementation of the standard library _sbrk() function.

Parameters

[in]	incr	The number of bytes being asked for by malloc().

Note

This function overrides the _sbrk version that exists in the newlib library that is linked with. That version improperly relies on the SP as part of it's allocation strategy. This is bad in general and worse in an RTOS environment. This version insures that we allocate the byte pool requested by malloc() only from our allocated HEAP area. Also note that newlib is pre-built and forces the pagesize used by malloc() to be 4096. That requires that we have a HEAP of at least 4096 if we are to support malloc().

Return values

Address	of allocated area if successful, -1 otherwise.

< Defined by the linker.

< Defined by the linker.

Need to align heap to word boundary, else will get

hard faults on Cortex-M0. So we assume that heap starts on

word boundary, hence make sure we always add a multiple of

4 to it.

< align value to 4

Heap has overflowed

Chapter 6 Structure Index

This section lists SSP structures.

6.1 Data Structures

This section includes SSP structures and members.

RBLE_GATT_RELIABLE_WRITE	Reliable Structure
RBLE_GATT_SET_DATA	Set Data Structure
RBLE_GATT_SET_PERM	Set Permission Structure
RBLE_GATT_SVC_128_LIST	Data list for service result - 128bit
RBLE_GATT_SVC_LIST	Data list for service result
RBLE_GATT_SVC_RANGE_LIST	Service range
RBLE_GATT_UUID_TYPE	UUID with different length Structure
RBLE_GATT_WRITE_CHAR_REQ	Write Characteristic Request Structure
RBLE_GATT_WRITE_RELIABLE_REQ	Write Reliable Characteristic Request Structure
RBLE_GATT_WRITE_RESP	Write Response Structure
riic_extended_cfg	
riic_instance_ctrl_t	
riic_slave_instance_ctrl_t	
rsa_api_t	
rsa_cfg_t	
rsa_ctrl_t	
rsa_instance_t	
rsa_key_t	
rspi_access_delay_t	
rspi_clock_delay_t	
rspi_instance_ctrl_t	
rspi_loopback_t	
rspi_mosi_idle_t	
rspi_parity_t	
rspi_ssl_negation_delay_t	
rspi_ssl_polarity_t	
rtc_alarm_time_t	
rtc_api_t	
rtc_callback_args_t	
rtc_cfg_t	
rtc_error_adjustment_cfg_t	
rtc_error_adjustment_mode_cfg_t	
rtc_info_t	
rtc_instance_ctrl_t	

6.1.1 d1_device_synergy Struct Reference

#include <sf_tes_2d_drw_base.h>

Detailed Description

Device handle type definition for Synergy implementation.

The documentation for this struct was generated from the following file:

- sf_tes_2d_drw_base.h

6.1.2 NX_DES Struct Reference

#include <nx_des.h>

Detailed Description

NetX Component DES Encryption Standard (DES)

The documentation for this struct was generated from the following file:

- nx_des.h

6.1.3 NX_IPV6_HEADER Struct Reference

#include <nx_ipv6.h>

Detailed Description

NetX Component Internet Protocol version 6 (IPv6)

The documentation for this struct was generated from the following file:

- nx_ipv6.h

6.1.4 NX_MD5 Struct Reference

#include <nx_md5.h>

Detailed Description

NetX Component MD5 Digest Algorithm (MD5)

The documentation for this struct was generated from the following file:

- nx_md5.h

6.1.5 NX_SECURE_TLS_PHASH_SCE Struct Reference

#include <nx_crypto_phash_sce.h>

Detailed Description

NetX Secure Component Transport Layer Security (TLS)

The documentation for this struct was generated from the following file:

- nx_crypto_phash_sce.h

6.1.6 NX_SECURE_TLS_PRF_1_SCE Struct Reference

#include <nx_crypto_tls_prf_1_sce.h>

Detailed Description

NetX Secure Component Transport Layer Security (TLS)

The documentation for this struct was generated from the following file:

- nx_crypto_tls_prf_1_sce.h

6.1.7 NX_SECURE_TLS_PRF_SHA_256_SCE Struct Reference

#include <nx_crypto_tls_prf_sha256_sce.h>

Detailed Description

NetX Secure Component Transport Layer Security (TLS)

The documentation for this struct was generated from the following file:

- nx_crypto_tls_prf_sha256_sce.h

6.1.8 NX_SHA1 Struct Reference

#include <nx_sha1.h>

Detailed Description

NetX Component SHA1 Digest Algorithm (SHA1)

The documentation for this struct was generated from the following file:

- nx_sha1.h

6.1.9 RBLE_GATT_CHAR_128_LIST Struct Reference

Data list for characteristic result - 128bit. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint8_t	prop
uint16_t	pointer_hdl
uint8_t	uuid [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

Data list for characteristic result - 128bit.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_CHAR_128_LIST::attr_hdl
database element handle

◆ pointer_hdl

uint16_t RBLE_GATT_CHAR_128_LIST::pointer_hdl
pointer handle to UUID

◆ prop

uint8_t RBLE_GATT_CHAR_128_LIST::prop
properties

◆ uuid

uint8_t RBLE_GATT_CHAR_128_LIST::uuid[RBLE_GATT_128BIT_UUID_OCTET]
characteristic UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.10 RBLE_GATT_CHAR_DESC_128_LIST Struct Reference

Special data list for descriptor result - 128bit. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint8_t	uuid [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

Special data list for descriptor result - 128bit.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_CHAR_DESC_128_LIST::attr_hdl
database element handle

◆ uuid

uint8_t RBLE_GATT_CHAR_DESC_128_LIST::uuid[RBLE_GATT_128BIT_UUID_OCTET]
128-bit descriptor UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.11 RBLE_GATT_CHAR_DESC_LIST Struct Reference

Special data list for descriptor result. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint16_t	desc_hdl

Detailed Description

Special data list for descriptor result.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_CHAR_DESC_LIST::attr_hdl
database element handle

◆ desc_hdl

uint16_t RBLE_GATT_CHAR_DESC_LIST::desc_hdl
descriptor UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.12 RBLE_GATT_CHAR_LIST Struct Reference

Data list for characteristic result. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint8_t	prop
uint16_t	pointer_hdl
uint16_t	uuid

Detailed Description

Data list for characteristic result.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_CHAR_LIST::attr_hdl
database element handle

◆ pointer_hdl

uint16_t RBLE_GATT_CHAR_LIST::pointer_hdl
pointer handle to UUID

◆ prop

uint8_t RBLE_GATT_CHAR_LIST::prop
properties

◆ uuid

uint16_t RBLE_GATT_CHAR_LIST::uuid
characteristic UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.13 RBLE_GATT_DESIRED_TYPE Struct Reference

desired UUID More...

#include <rble_api.h>

Data Fields

uint16_t	value_size
uint8_t	value [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

desired UUID

Field Documentation

◆ value

uint8_t RBLE_GATT_DESIRED_TYPE::value[RBLE_GATT_128BIT_UUID_OCTET]
actual UUID

◆ value_size

uint16_t RBLE_GATT_DESIRED_TYPE::value_size
Size of the UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.14 RBLE_GATT_DISC_CHAR_DESC_REQ Struct Reference

Characteristic Descriptor Discovery Request Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	conhdl
uint16_t	start_hdl
uint16_t	end_hdl

Detailed Description

Characteristic Descriptor Discovery Request Structure.

Parameters for RBLE_GATT_Discovery_Char_Descriptor_Request

Field Documentation

◆ conhdl

uint16_t RBLE_GATT_DISC_CHAR_DESC_REQ::conhdl
connection handle

◆ end_hdl

uint16_t RBLE_GATT_DISC_CHAR_DESC_REQ::end_hdl
end handle range

◆ start_hdl

uint16_t RBLE_GATT_DISC_CHAR_DESC_REQ::start_hdl
start handle range

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.15 RBLE_GATT_DISC_CHAR_REQ Struct Reference

Characteristic Discovery Request Structure. More...

#include <rble_api.h>

Data Fields

uint8_t	req_type
uint16_t	conhdl
uint16_t	start_hdl
uint16_t	end_hdl
RBLE_GATT_DESIRED_TYPE	desired_char

Detailed Description

Characteristic Discovery Request Structure.

Parameters for RBLE_GATT_Discovery_Char_Request

Field Documentation

◆ conhdl

uint16_t RBLE_GATT_DISC_CHAR_REQ::conhdl
connection handle

◆ desired_char

RBLE_GATT_DESIRED_TYPE RBLE_GATT_DISC_CHAR_REQ::desired_char
desired UUID in disc service char

◆ end_hdl

uint16_t RBLE_GATT_DISC_CHAR_REQ::end_hdl
end handle range

◆ req_type

uint8_t RBLE_GATT_DISC_CHAR_REQ::req_type
GATT request type

◆ start_hdl

uint16_t RBLE_GATT_DISC_CHAR_REQ::start_hdl
start handle range

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.16 RBLE_GATT_DISC_SVC_REQ Struct Reference

Service Discovery Request Structure. More...

#include <rble_api.h>

Data Fields

uint8_t	req_type
uint16_t	conhdl
uint16_t	start_hdl
uint16_t	end_hdl
RBLE_GATT_DESIRED_TYPE	desired_svc

Detailed Description

Service Discovery Request Structure.

Parameters for RBLE_GATT_Discovery_Service_Request

Field Documentation

◆ conhdl

uint16_t RBLE_GATT_DISC_SVC_REQ::conhdl
connection handle

◆ desired_svc

RBLE_GATT_DESIRED_TYPE RBLE_GATT_DISC_SVC_REQ::desired_svc
desired service: if 0x0000, discover all

◆ end_hdl

uint16_t RBLE_GATT_DISC_SVC_REQ::end_hdl
end handle range

◆ req_type

uint8_t RBLE_GATT_DISC_SVC_REQ::req_type
GATT request type

◆ start_hdl

uint16_t RBLE_GATT_DISC_SVC_REQ::start_hdl
start handle range

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.17 RBLE_GATT_EVENT Struct Reference

rBLE GATT Event Structure More...

#include <rble_api.h>

Data Fields

RBLE_GATT_EVENT_TYPE	type

Detailed Description

rBLE GATT Event Structure

Field Documentation

◆ type

RBLE_GATT_EVENT_TYPE RBLE_GATT_EVENT::type
type of GATT event

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.18 RBLE_GATT_EXE_WR_CHAR_REQ Struct Reference

execute write characteristic request Structure More...

#include <rble_api.h>

Data Fields

uint8_t	exe_wr_ena
uint16_t	conhdl

Detailed Description

execute write characteristic request Structure

Parameters for RBLE_GATT_Execute_Write_Char_Request

Field Documentation

◆ conhdl

uint16_t RBLE_GATT_EXE_WR_CHAR_REQ::conhdl
connection handle

◆ exe_wr_ena

uint8_t RBLE_GATT_EXE_WR_CHAR_REQ::exe_wr_ena
flag to describe if write or cancel

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.19 RBLE_GATT_INCL_128_LIST Struct Reference

Special data list for include result - 128bit. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint16_t	start_hdl
uint16_t	end_hdl
uint8_t	uuid [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

Special data list for include result - 128bit.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_INCL_128_LIST::attr_hdl
element handle

◆ end_hdl

uint16_t RBLE_GATT_INCL_128_LIST::end_hdl
end handle

◆ start_hdl

uint16_t RBLE_GATT_INCL_128_LIST::start_hdl
start handle

◆ uuid

uint8_t RBLE_GATT_INCL_128_LIST::uuid[RBLE_GATT_128BIT_UUID_OCTET]
included 128-bit service UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.20 RBLE_GATT_INCL_LIST Struct Reference

Data list for include result. More...

#include <rble_api.h>

Data Fields

uint16_t	attr_hdl
uint16_t	start_hdl
uint16_t	end_hdl
uint16_t	uuid

Detailed Description

Data list for include result.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_INCL_LIST::attr_hdl
element handle

◆ end_hdl

uint16_t RBLE_GATT_INCL_LIST::end_hdl
end handle

◆ start_hdl

uint16_t RBLE_GATT_INCL_LIST::start_hdl
start handle

◆ uuid

uint16_t RBLE_GATT_INCL_LIST::uuid
included service UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.21 RBLE_GATT_INDICATE_REQ Struct Reference

Indicate Request Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	conhdl
uint16_t	charhdl

Detailed Description

Indicate Request Structure.

Parameters for RBLE_GATT_Indicate_Request

Field Documentation

◆ charhdl

uint16_t RBLE_GATT_INDICATE_REQ::charhdl
characteristic handle

◆ conhdl

uint16_t RBLE_GATT_INDICATE_REQ::conhdl
connection handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.22 RBLE_GATT_INFO_DATA Struct Reference

Attribute data holder. More...

#include <rble_api.h>

Data Fields

uint8_t	each_len
uint8_t	len
uint8_t	data [RBLE_GATT_MAX_VALUE]

Detailed Description

Attribute data holder.

Field Documentation

◆ data

uint8_t RBLE_GATT_INFO_DATA::data[RBLE_GATT_MAX_VALUE]
data

◆ each_len

uint8_t RBLE_GATT_INFO_DATA::each_len
each result length

◆ len

uint8_t RBLE_GATT_INFO_DATA::len
data length

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.23 RBLE_GATT_NOTIFY_REQ Struct Reference

notify request Structure More...

#include <rble_api.h>

Data Fields

uint16_t	conhdl
uint16_t	charhdl

Detailed Description

notify request Structure

Parameters for RBLE_GATT_Notify_Request

Field Documentation

◆ charhdl

uint16_t RBLE_GATT_NOTIFY_REQ::charhdl
characteristic handle

◆ conhdl

uint16_t RBLE_GATT_NOTIFY_REQ::conhdl
connection handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.24 RBLE_GATT_QUERY_RESULT Struct Reference

Query result for multiple responses. More...

#include <rble_api.h>

Data Fields

uint8_t	len
uint8_t	value [RBLE_GATT_MAX_VALUE]

Detailed Description

Query result for multiple responses.

Field Documentation

◆ len

uint8_t RBLE_GATT_QUERY_RESULT::len
length of value

◆ value

uint8_t RBLE_GATT_QUERY_RESULT::value[RBLE_GATT_MAX_VALUE]
data result from query

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.25 RBLE_GATT_READ_CHAR_REQ Struct Reference

Read Characteristic Values and Descriptor Request Structure. More...

#include <rble_api.h>

Data Fields

uint8_t	req_type
uint16_t	offset
uint16_t	conhdl
uint16_t	start_hdl
uint16_t	end_hdl
uint16_t	nb_uuid
RBLE_GATT_UUID_TYPE	uuid [RBLE_GATT_MAX_NB_HDLS]

Detailed Description

Read Characteristic Values and Descriptor Request Structure.

Parameters for RBLE_GATT_Read_Char_Request

Field Documentation

◆ conhdl

uint16_t RBLE_GATT_READ_CHAR_REQ::conhdl
connection handle

◆ end_hdl

uint16_t RBLE_GATT_READ_CHAR_REQ::end_hdl
end handle range

◆ nb_uuid

uint16_t RBLE_GATT_READ_CHAR_REQ::nb_uuid
number of UUID

◆ offset

uint16_t RBLE_GATT_READ_CHAR_REQ::offset
read offset

◆ req_type

uint8_t RBLE_GATT_READ_CHAR_REQ::req_type
request type

◆ start_hdl

uint16_t RBLE_GATT_READ_CHAR_REQ::start_hdl
start handle range

◆ uuid

RBLE_GATT_UUID_TYPE RBLE_GATT_READ_CHAR_REQ::uuid[RBLE_GATT_MAX_NB_HDLS]
characteristic UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.26 RBLE_GATT_RELIABLE_WRITE Struct Reference

Reliable Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	elmt_hdl
uint16_t	size
uint8_t	value [RBLE_GATT_MAX_RELIABLE_WRITE_CONTENTS]

Detailed Description

Reliable Structure.

Field Documentation

◆ elmt_hdl

uint16_t RBLE_GATT_RELIABLE_WRITE::elmt_hdl
characteristic handle

◆ size

uint16_t RBLE_GATT_RELIABLE_WRITE::size
value size

◆ value

uint8_t RBLE_GATT_RELIABLE_WRITE::value[RBLE_GATT_MAX_RELIABLE_WRITE_CONTENTS]
value holder

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.27 RBLE_GATT_SET_DATA Struct Reference

Set Data Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	val_hdl
uint16_t	val_len
uint8_t	value [RBLE_GATT_MAX_LONG_VALUE]

Detailed Description

Set Data Structure.

Parameters for RBLE_GATT_Set_Data

Field Documentation

◆ val_hdl

uint16_t RBLE_GATT_SET_DATA::val_hdl
value handle

◆ val_len

uint16_t RBLE_GATT_SET_DATA::val_len
size of the value data

◆ value

uint8_t RBLE_GATT_SET_DATA::value[RBLE_GATT_MAX_LONG_VALUE]
value data

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.28 RBLE_GATT_SET_PERM Struct Reference

Set Permission Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	start_hdl
uint16_t	end_hdl
uint16_t	perm

Detailed Description

Set Permission Structure.

Parameters for RBLE_GATT_Set_Permission

Field Documentation

◆ end_hdl

uint16_t RBLE_GATT_SET_PERM::end_hdl
end handle range

◆ perm

uint16_t RBLE_GATT_SET_PERM::perm
Permission of attr

◆ start_hdl

uint16_t RBLE_GATT_SET_PERM::start_hdl
start handle range

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.29 RBLE_GATT_SVC_128_LIST Struct Reference

Data list for service result - 128bit. More...

#include <rble_api.h>

Data Fields

uint16_t	start_hdl
uint16_t	end_hdl
uint8_t	attr_hdl [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

Data list for service result - 128bit.

Field Documentation

◆ attr_hdl

uint8_t RBLE_GATT_SVC_128_LIST::attr_hdl[RBLE_GATT_128BIT_UUID_OCTET]
attribute handle

◆ end_hdl

uint16_t RBLE_GATT_SVC_128_LIST::end_hdl
end handle

◆ start_hdl

uint16_t RBLE_GATT_SVC_128_LIST::start_hdl
start handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.30 RBLE_GATT_SVC_LIST Struct Reference

Data list for service result. More...

#include <rble_api.h>

Data Fields

uint16_t	start_hdl
uint16_t	end_hdl
uint16_t	attr_hdl

Detailed Description

Data list for service result.

Field Documentation

◆ attr_hdl

uint16_t RBLE_GATT_SVC_LIST::attr_hdl
attribute handle

◆ end_hdl

uint16_t RBLE_GATT_SVC_LIST::end_hdl
end handle

◆ start_hdl

uint16_t RBLE_GATT_SVC_LIST::start_hdl
start handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.31 RBLE_GATT_SVC_RANGE_LIST Struct Reference

service range More...

#include <rble_api.h>

Data Fields

uint16_t	start_hdl
uint16_t	end_hdl

Detailed Description

service range

Field Documentation

◆ **end_hdl**

uint16_t RBLE_GATT_SVC_RANGE_LIST::end_hdl
end handle

◆ **start_hdl**

uint16_t RBLE_GATT_SVC_RANGE_LIST::start_hdl
start handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.32 RBLE_GATT_UUID_TYPE Struct Reference

UUID with different length Structure. More...

#include <rble_api.h>

Data Fields

uint8_t	value_size
uint8_t	expect_resp_size
uint8_t	value [RBLE_GATT_128BIT_UUID_OCTET]

Detailed Description

UUID with different length Structure.

Field Documentation

◆ **expect_resp_size**

uint8_t RBLE_GATT_UUID_TYPE::expect_resp_size
expected response size - read multiple

◆ value

uint8_t RBLE_GATT_UUID_TYPE::value[RBLE_GATT_128BIT_UUID_OCTET]
actual UUID

◆ value_size

uint8_t RBLE_GATT_UUID_TYPE::value_size
Size of the UUID

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.33 RBLE_GATT_WRITE_CHAR_REQ Struct Reference

Write Characteristic Request Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	conhdl
uint16_t	charhdl
uint16_t	wr_offset
uint16_t	val_len
uint8_t	req_type
uint8_t	auto_execute
uint8_t	value [RBLE_GATT_MAX_LONG_VALUE]

Detailed Description

Write Characteristic Request Structure.

Parameters for RBLE_GATT_Write_Char_Request

Field Documentation

◆ auto_execute

uint8_t RBLE_GATT_WRITE_CHAR_REQ::auto_execute
execute write

◆ charhdl

uint16_t RBLE_GATT_WRITE_CHAR_REQ::charhdl
valid characteristic handle

◆ conhdl

uint16_t RBLE_GATT_WRITE_CHAR_REQ::conhdl
connection handle

◆ req_type

uint8_t RBLE_GATT_WRITE_CHAR_REQ::req_type
request type

◆ val_len

uint16_t RBLE_GATT_WRITE_CHAR_REQ::val_len
size of the value data

◆ value

uint8_t RBLE_GATT_WRITE_CHAR_REQ::value[RBLE_GATT_MAX_LONG_VALUE]
check, maybe union type is required

◆ wr_offset

uint16_t RBLE_GATT_WRITE_CHAR_REQ::wr_offset
offset to write

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.34 RBLE_GATT_WRITE_RELIABLE_REQ Struct Reference

Write Reliable Characteristic Request Structure. More...

#include <rble_api.h>

Data Fields

uint8_t	nb_writes
uint8_t	auto_execute
uint16_t	conhdl
RBLE_GATT_RELIABLE_WRITE	value [RBLE_GATT_MAX_RELIABLE_WRITE_NUM]

Detailed Description

Write Reliable Characteristic Request Structure.

Parameters for RBLE_GATT_Write_Reliable_Request

Field Documentation

◆ auto_execute

uint8_t RBLE_GATT_WRITE_RELIABLE_REQ::auto_execute
automatic execute write

◆ conhdl

uint16_t RBLE_GATT_WRITE_RELIABLE_REQ::conhdl
connection handle

◆ nb_writes

uint8_t RBLE_GATT_WRITE_RELIABLE_REQ::nb_writes
number of reliable writes

◆ value

RBLE_GATT_RELIABLE_WRITE RBLE_GATT_WRITE_RELIABLE_REQ::value[RBLE_GATT_MAX_RELIABLE_WRITE_NUM]
number of reliable

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.35 RBLE_GATT_WRITE_RESP Struct Reference

Write Response Structure. More...

#include <rble_api.h>

Data Fields

uint16_t	conhdl
uint16_t	att_hdl
uint8_t	att_code

Detailed Description

Write Response Structure.

Parameters for RBLE_GATT_Write_Response

Field Documentation

◆ att_code

uint8_t RBLE_GATT_WRITE_RESP::att_code
ATT code

◆ att_hdl

uint16_t RBLE_GATT_WRITE_RESP::att_hdl
Attribute handle

◆ conhdl

uint16_t RBLE_GATT_WRITE_RESP::conhdl
Connection handle

The documentation for this struct was generated from the following file:

- rble_api.h

6.1.36 sdmmc_priv_csd_reg_ext_t Struct Reference

#include <sdcard.h>

Detailed Description

SDMMC card specific data extended

The documentation for this struct was generated from the following file:

- sdcard.h

6.1.37 sdmmc_priv_csd_reg_t Struct Reference

#include <sdcard.h>

Detailed Description

SDMMC card specific data

The documentation for this struct was generated from the following file:

- sdcard.h

6.1.38 sf_cellular_circular_queue_cfg_t Struct Reference

#include <sf_cellular_common_api.h>

Data Fields

uint32_t *	p_circular_queue_buffer Circular Queue buffer.
ULONG	queue_size Circular Queue Size.
uint8_t	ok_check_index Variable to store index for data checking success string response.
uint8_t	error_check_index Variable to store index for data checking error string response.
TX_QUEUE *	p_circular_queue Circular Queue.

Detailed Description

Circular queue configuration

The documentation for this struct was generated from the following file:

- sf_cellular_common_api.h

6.1.39 sf_cellular_comms_extend_cfg_t Struct Reference

#include <sf_cellular_common_api.h>

Data Fields

sf_comms_instance_t const *	p_sf_comms_instance Lower level HAL driver instance.
TX_THREAD *	p_sf_comms_rx_thread Received data ThreadX ID.
uint8_t *	p_sf_comms_rx_thread_stack Receive data thread stack memory pointer.
ULONG	sf_comms_rx_thread_stack_size Receive data thread stack size.
UINT	rx_thread_priority Receive data thread priority.
uint8_t	do_run Thread running status.

Detailed Description

SF Communication framework configuration

The documentation for this struct was generated from the following file:

- sf_cellular_common_api.h

6.1.40 sf_cellular_extended_cfg_t Struct Reference

#include <sf_cellular_common_api.h>

Data Fields

sf_cellular_circular_queue_cfg_t *	p_circular_queue_cfg Circular Queue configuration.
sf_cellular_comms_extend_cfg_t *	p_sf_comms_cfg Lower level HAL interface configuration.
ioport_port_pin_t	pin_reset Port pin used for resetting cellular module.
ioport_level_t	reset_level Module reset level.
void *	p_module_extended_cfg Instance specific module configuration.

Detailed Description

SF Cellular framework extended configuration

The documentation for this struct was generated from the following file:

- sf_cellular_common_api.h

6.1.41 sf_cellular_instance_cfg_t Struct Reference

#include <sf_cellular_common_api.h>

Data Fields

uint8_t	init_done Status flag storing driver initialization status.

uint8_t	is_opened Status flag storing framework open status.
uint8_t	is_data_mode_on Status flag storing data mode status.
sf_cellular_cfg_t const *	p_cfg Instance configuration.
sf_cellular_provisioning_t	prov_info Provisioning information.
sf_cellular_stats_t	celr_stats Cellular Statistics information.
TX_MUTEX *	p_cellular_mutex Mutex for Framework API synchronization.
uint8_t *	p_socket_status_buffer Buffer to store the socket status information.
uint16_t	socket_status_buffer_length Size of Socket status buffer.

Detailed Description

SF Cellular framework instance configuration

The documentation for this struct was generated from the following file:

- sf_cellular_common_api.h

6.1.42 sf_cellular_qctlcatm1_socket_cfg_t Struct Reference

#include <sf_cellular_qctlcatm1_socket_private_api.h>

Data Fields

uint16_t	transpktsize
uint16_t	max_backoffs
uint16_t	max_rto

Detailed Description

Structure definition for socket parameter configuration

Field Documentation

◆ max_backoffs

uint16_t sf_cellular_qctlcatm1_socket_cfg_t::max_backoffs
Maximum Number of Retransmissions. The range is 3-20, and the default value is 8

◆ max_rto

uint16_t sf_cellular_qctlcatm1_socket_cfg_t::max_rto
The maximum interval time of TCP retransmission. The range is 5-1000, and the default value is 600. Unit: 100ms

◆ transpktsize

uint16_t sf_cellular_qctlcatm1_socket_cfg_t::transpktsize
The max length of the data packet to be sent. The range is 1 -1460, and the default value is 1024. Unit: byte

---

The documentation for this struct was generated from the following file:

- sf_cellular_qctlcatm1_socket_private_api.h

6.1.43 sf_cellular_socket_info_t Struct Reference

#include <sf_cellular_ryz014catm1_socket_private_api.h>

Data Fields

sf_cellular_qctlcatm1_socket_state_t	state Socket state.
uint8_t	local_ip [SF_CELLULAR_STR_LEN_16] Local IP address to which socket is bind.
uint16_t	local_port Local port to which socket is bind.
uint8_t	remote_ip [SF_CELLULAR_STR_LEN_16] Remote IP to which socket is connected. More...
uint16_t	remote_port Remote port with which socket is connected.
sf_cellular_qctlcatm1_socket_type_t	sock_type Socket type.
uint16_t	sin_port Port number to which socket need to bind.
uint8_t	data_state Data Received notification.
int8_t	new_conn_fd Socket descriptor of new incoming connection.
sf_cellular_ryz014catm1_socket_state_t	state Socket state.
sf_cellular_ryz014catm1_soc	sock_type

ket_type_t	Socket type.
uint16_t	timeout Timeout period.
uint16_t	pckt_size Packet size to be used.
uint16_t	tx_timeout Data sending timeout.
uint16_t	conn_timeout Connection timeout.

Detailed Description

Structure definition for socket details information

Field Documentation

◆ remote_ip

uint8_t sf_cellular_socket_info_t::remote_ip
Remote IP to which socket is connected. Remote ip to which socket is connected.

The documentation for this struct was generated from the following files:

- sf_cellular_qctlcatm1_socket_private_api.h
- sf_cellular_ryz014catm1_socket_private_api.h

6.1.44 ssp_pack_version_t Union Reference

#include <ssp_version.h>

Data Fields

uint32_t	version_id
struct {	
uint8_t build	
	Build version of SSP Pack.
uint8_t patch	
	Patch version of SSP Pack.
uint8_t minor	
	Minor version of SSP Pack.
uint8_t major	
	Major version of SSP Pack.
};	

Detailed Description

SSP Pack version structure

Field Documentation

◆ @42

struct { ... }
Code version parameters, little endian order.

◆ version_id

uint32_t ssp_pack_version_t::version_id
Version id

The documentation for this union was generated from the following file:

- ssp_version.h

6.1.45 ssp_version_t Union Reference

#include <ssp_common_api.h>

Data Fields

uint32_t	version_id
struct {	
uint8_t code_version_minor	
	Code minor version.
uint8_t code_version_major	
	Code major version.
uint8_t api_version_minor	
	API minor version.
uint8_t api_version_major	
	API major version.
};	

Detailed Description

Common version structure

Field Documentation

◆ **@38**

struct { ... }
Code version parameters

◆ version_id

uint32_t ssp_version_t::version_id
Version id

The documentation for this union was generated from the following file:

- ssp_common_api.h

6.2 Data Structure Index

a | b | c | d | e | f | g | h | i | j | k | l | n | o | p | q | r | s | t | u | w

a

b

C

d

e

f

g

h

i

j

k

l

r

S

t

u

w

6.3 Data Fields

This section lists SSP data fields.

6.3.1 All Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- a -

- a : display_color_t
- absolute_start_addr : sf_el_lx_nor_memory_settings_t
- accept : sf_ble_bonding_response_t
- accept_addr : sf_ble_addr_verify_ind_t
- access_control : sf_wifi_cfg_t
- access_delay : spi_on_rspi_cfg_t
- access_ipl : sdmmc_cfg_t
- access_tech_name : sf_cellular_network_status_t

- accessWindowClear : flash_api_t
- accessWindowSet : flash_api_t
- accuracy : sf_ble_cts_ref_time_t
- ACLAdd : sf_wifi_api_t
- ACLDelete : sf_wifi_api_t
- activation_on_rxi : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- activation_on_txi : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- activation_source : transfer_cfg_t
- active : sf_thread_monitor_thread_counter_t
- active_band : sf_cellular_network_status_t
- actual_count : sf_el_fx_media_partition_info_t
- actual_hwErr_event : riic_instance_ctrl_t
- ad_da_synchronized : dac_cfg_t
- adc_calib_available : adc_instance_ctrl_t
- add_average_count : adc_cfg_t
- add_mask : adc_channel_cfg_t
- addAdditionalAuthenticationData : aes_api_t
- addition_supported : bsp_feature_adc_t
- addr : sf_ble_addr_t , sf_wifi_ip_addr_t , sf_wifi_qca4010_ip_addr_t
- addr_high : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_loaded : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_low : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_mode : i2c_cfg_t
- addr_remain : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_total : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_type : sf_ble_addr_verify_ind_t , sf_ble_connection_t , sf_ble_scan_t
- address : ptp_instance_ctrl_t
- address_restarted : riic_instance_ctrl_t
- address_type : sf_ble_scan_info_t
- adjust_reason : sf_ble_prf_cts_curr_time_t
- adjustment_mode : rtc_error_adjustment_mode_cfg_t
- adjustment_period : rtc_error_adjustment_mode_cfg_t
- adjustment_type : rtc_error_adjustment_cfg_t
- adjustment_value : rtc_error_adjustment_cfg_t
- adv_chnl_map : sf_ble_adv_info_t
- adv_data : sf_ble_adv_info_t
- adv_data_length : sf_ble_adv_data_t
- adv_filt_policy : sf_ble_adv_info_t
- adv_intv_max : sf_ble_adv_info_t
- adv_intv_min : sf_ble_adv_info_t
- adv_type : sf_ble_adv_info_t
- advertisementStart : sf_ble_api_t
- advertisementStop : sf_ble_api_t
- agt_link : opamp_on_opamp_cfg_t
- agtio_output_enabled : timer_on_agt_cfg_t
- agto_output_enabled : timer_on_agt_cfg_t
- agtoa_output_enable : timer_on_agt_cfg_t
- agtob_output_enable : timer_on_agt_cfg_t
- airplane_mode : sf_cellular_provisioning_t
- alarm_ipl : rtc_cfg_t
- alarm_irq : rtc_instance_ctrl_t
- alert : sf_ble_anp_api_unread_alert_ntf_t
- alert_lvl : sf_ble_prf_ias_alert_lvl_change_t
- alert_num : sf_ble_anp_api_new_alert_t
- alert_status : sf_ble_prf_alert_status_ntf_t

- aligned_buff : sdmmc_instance_ctrl_t
- alignment : adc_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t
- alpha_value : jpeg_decode_cfg_t
- api_version_major : ssp_version_t
- api_version_minor : ssp_version_t
- apn : sf_cellular_provisioning_t
- arc4Process : arc4_api_t
- argumentFind : sf_console_api_t
- att_code : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Cancel_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Discovery_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Reliable_Resp_t , RBLE_GATT_WRITE_RESP
- att_hdl : RBLE_GATT_WRITE_RESP
- attr_declare_handle : sf_ble_char_attribute_t
- attr_declare_type : sf_ble_char_attribute_t
- attr_handle : sf_ble_gatt_attr_event_t , sf_ble_svc_attribute_t
- attr_hdl : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_DESC_128_LIST , RBLE_GATT_CHAR_DESC_LIST , RBLE_GATT_CHAR_LIST , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , sf_ble_long_attr_info_t
- attr_perm : sf_ble_char_attribute_t
- attr_properties : sf_ble_char_attribute_t
- attr_type : sf_ble_svc_attribute_t
- attr_uuid : sf_ble_char_attribute_t
- attr_value_handle : sf_ble_char_attribute_t
- attr_value_len : sf_ble_char_attribute_t , sf_ble_svc_attribute_t
- atune1 : ctsu_cfg_t
- audio_clk_freq_hz : i2s_cfg_t
- audio_clock : i2s_on_ssi_cfg_t
- auth_req : sf_ble_bonding_req_ind_t
- auth_type : sf_ble_sec_enc_start_ind_t , sf_cellular_nsal_cfg_t , sf_cellular_provisioning_t
- authorization : sf_ble_api_t
- auto_enable : transfer_cfg_t
- auto_execute : RBLE_GATT_WRITE_CHAR_REQ , RBLE_GATT_WRITE_RELIABLE_REQ , sf_ble_char_write_req_t
- autoClearEvent : ptp_api_t
- autostart : elc_cfg_t , external_irq_cfg_t , input_capture_cfg_t , keymatrix_cfg_t , sf_console_cfg_t , timer_cfg_t , wdt_cfg_t
- available : sf_comms_telnet_instance_ctrl_t

- average : ctsu_instance_ctrl_t , sdadc_channel_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- b -

- b : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- back_porch : display_timing_t , glcd_ctrl_t
- bankSelect : qspi_api_t
- base_addr : sf_el_fx_media_ebr_info_t
- batt_lvl : sf_ble_bas_battery_lvl_ntf_t
- baud_rate : uart_cfg_t
- baud_rate_error_x_1000 : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- baud_rate_prescaler : can_bit_timing_cfg_t
- baudclk_out : uart_on_sci_cfg_t
- baudSet : uart_api_t
- bcast_mode : sf_ble_provisioning_t
- bclk_div : cgc_system_clock_cfg_t
- bd_addr : sf_ble_addr_verify_ind_t , sf_ble_bonding_req_ind_t , sf_ble_cfg_t , sf_ble_chipset_info_t , sf_ble_connection_t , sf_ble_scan_t
- bd_buffer_ptr : EMAC_BD
- bd_bufsize : EMAC_BD
- bd_nx_packet : EMAC_BD
- bd_rxdatalength : EMAC_BD
- bd_status : EMAC_BD
- beacon : sf_wifi_cfg_t
- ber : sf_cellular_info_t
- bg_color : display_layer_t , display_output_cfg_t
- bias_method : slcdc_cfg_t
- binfo : sf_touch_ctsu_instance_ctrl_t
- bit_order : crc_cfg_t , crc_instance_ctrl_t , spi_cfg_t
- bit_width : dac_info_t
- bitrate : spi_cfg_t
- bitrate_modulation : sci_i2c_extended_cfg , sci_spi_extended_cfg , sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- blankCheck : flash_api_t
- ble_driver_thread_priority : sf_ble_on_rl78g1d_cfg_t
- ble_prf_value : sf_ble_prf_hid_change_event_t
- ble_serial_thread_priority : sf_ble_on_rl78g1d_cfg_t
- block_pool : sf_message_instance_ctrl_t
- block_section_end_addr : flash_fmi_block_info_t
- block_section_st_addr : flash_fmi_block_info_t
- block_size : flash_fmi_block_info_t , sdmmc_extended_cfg_t , sf_block_media_cfg_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_block_media_ram_instance_ctrl_t , sf_block_media_sdmmc_instance_ctrl_t
- block_size_write : flash_fmi_block_info_t
- blockReset : transfer_api_t
- bonding_mode : sf_ble_provisioning_t
- bonding_status : sf_ble_sec_enc_start_ind_t
- bondingResponse : sf_ble_api_t

- bondingStart : sf_ble_api_t
- boot_record_table : sf_el_fx_media_info_t
- brightness : display_correction_t , display_output_cfg_t
- BSP_ALIGN_VARIABLE_V2() : ptpedmac_instance_ctrl_t , sf_audio_playback_common_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t
- bss_type : sf_wifi_scan_t
- bssid : sf_wifi_qca4010_scan_t , sf_wifi_scan_t
- buff : sf_el_fx_media_ebr_info_t , sf_el_fx_media_mbr_info_t
- buff_len : sf_cellular_cmd_resp_t , sf_wifi_qca4010_cmd_resp_t
- buffer : sf_crypto_signature_context_t
- buffer_index : sf_adc_periodic_callback_args_t , sf_audio_playback_common_instance_ctrl_t
- buffer_keep : sf_message_acquire_cfg_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- buffer_size : sf_audio_record_i2s_instance_ctrl_t , sf_message_cfg_t , sf_message_instance_ctrl_t
- bufferAcquire : sf_message_api_t
- bufferRelease : sf_message_api_t
- build : ssp_pack_version_t
- bus_width : sdmmc_hw_t , sdmmc_info_t
- busClockOutCfg : cgc_api_t
- busClockOutDisable : cgc_api_t
- busClockOutEnable : cgc_api_t
- byte_pool : sf_crypto_instance_ctrl_t
- byte_swap : spi_on_rspi_cfg_t
- bytes : i2c_callback_args_t , sf_console_callback_args_t
- bytes_per_pixel : pdc_cfg_t , pdc_instance_ctrl_t , sf_el_gx_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- c -

- cac_api_open : cac_instance_ctrl_t
- cac_continous_mode : cac_instance_ctrl_t
- cac_lock : cac_instance_ctrl_t
- cac_lower_limit : cac_cfg_t
- cac_meas_clock : cac_cfg_t
- cac_ref_clock : cac_cfg_t
- cac_upper_limit : cac_cfg_t
- cache_state : flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t
- calculate : crc_api_t
- calendarAlarmGet : rtc_api_t
- calendarAlarmSet : rtc_api_t
- calendarCounterStart : rtc_api_t
- calendarCounterStop : rtc_api_t
- calendarTimeGet : rtc_api_t
- calendarTimeSet : rtc_api_t
- calib_adc_skip : adc_cfg_t
- calib_end_irq : sdadc_instance_ctrl_t
- calib_status : sdadc_instance_ctrl_t
- calibrate : adc_api_t , sf_touch_panel_v2_api_t , sf_touch_panel_v2_instance_ctrl_t

- calibration_data : adc_info_t
- calibration_end_ipl : adc_on_sdadc_cfg_t
- calibration_ongoing : adc_info_t
- calibration_reg_available : bsp_feature_adc_t
- callback : adc_instance_ctrl_t , sf_console_command_t
- callback_used : sf_external_irq_instance_ctrl_t
- callbackSet : ctsu_api_t , sf_touch_ctsu_api_t
- cancel_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- canvasInit : sf_el_gx_api_t
- cap : ctsu_cfg_t
- capture_count : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- capture_data_buffer_size : sf_audio_record_cfg_t
- capture_data_size : sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t
- capture_irq : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- capture_irq_ipl : input_capture_cfg_t
- captureStart : pdc_api_t
- card_detect : sdmmc_extended_cfg_t
- card_ipl : sdmmc_cfg_t
- card_type : sdmmc_info_t
- carry_ipl : rtc_cfg_t
- carry_irq : rtc_instance_ctrl_t
- carry_isr_triggered : rtc_instance_ctrl_t
- category_id : sf_ble_anp_ancp_t , sf_ble_anp_api_new_alert_t , sf_ble_anp_api_unread_alert_t
- cccd_val : sf_ble_onboard_profile_cccd_changed_t
- celr_stats : sf_cellular_instance_cfg_t
- cf_block_size_write : bsp_feature_flash_hp
- cf_macro_size : bsp_feature_flash_lp
- cfgGet : wdt_api_t
- chain_mode : transfer_info_t
- channel : adc_callback_args_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , can_callback_args_t , can_cfg_t , can_instance_ctrl_t , comparator_callback_args_t , comparator_cfg_t , dac8_instance_ctrl_t , dac_cfg_t , dac_instance_ctrl_t , dmac_instance_ctrl_t , external_irq_callback_args_t , external_irq_cfg_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_cfg_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_callback_args_t , input_capture_cfg_t , NX_REC , opamp_trim_args_t , ptpedmac_callback_args_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_calibrate_args_t , sdmmc_hw_t , sf_i2c_bus_t , sf_spi_bus_t , sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_qca4010_status_t , sf_wifi_scan_t , spi_callback_args_t , spi_cfg_t , ssi_instance_ctrl_t , timer_cfg_t , transfer_on_dmac_cfg_t , uart_callback_args_t , uart_cfg_t
- channel_opened : dac8_instance_ctrl_t , dac_instance_ctrl_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t
- channel_started : dac8_instance_ctrl_t , dac_instance_ctrl_t
- channels : keymatrix_callback_args_t , keymatrix_cfg_t , kint_instance_ctrl_t
- char_code : sf_ble_blp_meas_recv_data_t , sf_ble_onboard_profile_cccd_changed_t
- char_handle : sf_ble_char_discovery_rsp_t
- char_read_type : sf_ble_char_read_req_t
- char_write_type : sf_ble_char_write_req_t
- charhdl : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_INDICATE_REQ , RBLE_GATT_NOTIFY_REQ , RBLE_GATT_WRITE_CHAR_REQ
- check_pclkb_ratio : bsp_feature_can_t
- checkINFABTstatus : ptp_api_t

- checkWorst10Values : ptp_api_t
- chip_select : sf_spi_cfg_t , sf_spi_instance_ctrl_t
- chip_select_level_active : sf_spi_cfg_t , sf_spi_instance_ctrl_t
- chipset : sf_ble_info_t , sf_cellular_info_t
- CHNE : dtc_reg_t
- CHNS : dtc_reg_t
- cid : sf_cellular_network_status_t
- cipher_algorithm_type : sf_crypto_cipher_instance_ctrl_t
- cipher_chaining_mode : sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t
- cipherAadUpdate : sf_crypto_cipher_api_t
- cipherFinal : sf_crypto_cipher_api_t
- cipherInit : sf_crypto_cipher_api_t
- cipherUpdate : sf_crypto_cipher_api_t
- class_code : sf_message_header_t
- class_instance : sf_audio_playback_cfg_t , sf_message_header_t , sf_audio_playback_instance_ctrl_t
- clearINFABTstatus : ptp_api_t
- clearing : adc_cfg_t
- clearIOKeep : lpmv2_api_t
- clk_accuracy : sf_ble_connect_info_t
- clk_phase : spi_cfg_t
- clk_polarity : spi_cfg_t
- clk_src : uart_on_sci_cfg_t
- clksrc : glcd_cfg_t
- clock : bsp_feature_can_t , bsp_feature_rspi_t , bsp_feature_sci_t , cac_meas_clock_config_t , cac_ref_clock_config_t
- clock_delay : spi_on_rspi_cfg_t
- clock_div_ratio : glcd_cfg_t
- clock_divider : agt_input_capture_extend_t , gpt_input_capture_extend_t
- clock_division : pdc_cfg_t , wdt_cfg_t
- clock_frequency : timer_info_t
- clock_frequency_hz : wdt_timeout_values_t
- clock_rate : sdmmc_info_t
- clock_source : bsp_feature_adc_t , can_extended_cfg_t , can_instance_ctrl_t , rtc_cfg_t , rtc_info_t , rtc_instance_ctrl_t
- clockCheck : cgc_api_t
- clockOutCfg : cgc_api_t
- clockOutDisable : cgc_api_t
- clockOutEnable : cgc_api_t
- clocksCfg : cgc_api_t
- clockStart : cgc_api_t
- clockStop : cgc_api_t
- close : adc_api_t , aes_api_t , arc4_api_t , cac_api_t , can_api_t , comparator_api_t , crc_api_t , crypto_api_t , ctsu_api_t , dac_api_t , display_api_t , doc_api_t , dsa_api_t , ecc_api_t , external_irq_api_t , flash_api_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , input_capture_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , lvd_api_t , opamp_api_t , pdc_api_t , ptp_api_t , ptpedmac_api_t , qspi_api_t , rsa_api_t , rtc_api_t , sdmmc_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t , sf_el_gx_api_t , sf_external_irq_api_t , sf_i2c_api_t , sf_jpeg_decode_api_t , sf_memory_api_t , sf_message_api_t , sf_power_profiles_v2_api_t ,

sf_socket_api_t , sf_spi_api_t , sf_thread_monitor_api_t , sf_touch_ctsu_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t

- close_option : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- clut : display_api_t
- cmd_index : sf_cellular_command_parameters_info_t
- code : sf_message_header_t
- code_flash : flash_info_t
- code_version_major : ssp_version_t
- code_version_minor : ssp_version_t
- coefficient_m : sdadc_channel_cfg_t
- coefficient_n : sdadc_channel_cfg_t
- color_num : display_clut_t
- color_order : display_output_cfg_t
- color_space : jpeg_decode_cfg_t
- command : sf_console_command_t
- command_flag : sf_wifi_qca4010_instance_cfg_t
- command_id : sf_ble_anp_ancp_t
- command_list : sf_console_menu_t
- commandSend : sf_cellular_api_t
- CommandSend : sf_wifi_qca4010_api_t
- common_instance_mutex : sf_audio_playback_common_instance_ctrl_t
- communicationAbort : uart_api_t
- con_interval : sf_ble_cfg_t , sf_ble_connect_info_t
- con_latency : sf_ble_connect_info_t
- configure : ptp_api_t , rtc_api_t
- conhdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Cancel_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Discovery_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Reliable_Resp_t , RBLE_GATT_EXE_WR_CHAR_REQ , RBLE_GATT_INDICATE_REQ , RBLE_GATT_NOTIFY_REQ , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_WRITE_CHAR_REQ ,

RBLE_GATT_WRITE_RELIABLE_REQ , RBLE_GATT_WRITE_RESP , sf_ble_anp_ancp_change_t , sf_ble_anp_api_new_alert_ntf_t , sf_ble_anp_api_unread_alert_ntf_t , sf_ble_bas_battery_lvl_ntf_t , sf_ble_blp_meas_recv_data_t , sf_ble_connect_info_t , sf_ble_cts_curr_time_ntf_t , sf_ble_disconnect_t , sf_ble_hrp_api_meas_ntf_t , sf_ble_hrp_cp_change_t , sf_ble_prf_alert_status_ntf_t , sf_ble_prf_hid_change_event_t , sf_ble_prf_hid_report_ind_t , sf_ble_prf_htp_temp_info_ind_t , sf_ble_prf_ias_alert_lvl_change_t , sf_ble_prf_ringer_cp_change_t , sf_ble_prf_ringer_setting_ntf_t , sf_ble_scps_scan_intv_change_t , sf_ble_sec_enc_start_ind_t , sf_ble_sm_tk_info_t , sf_ble_tip_cp_change_t , st_sf_ble_prf_htp_meas_intv_val_t

- conn_handle : sf_ble_onboard_profile_cccd_changed_t
- conn_idx : sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- conn_mode : sf_ble_adv_info_t
- conn_timeout : sf_cellular_socket_info_t
- connect : analog_connect_api_t , sf_ble_api_t
- connectMultiple : analog_connect_api_t
- context : sf_console_callback_args_t , sf_console_command_t
- context_id : sf_cellular_provisioning_t
- contextInit : sf_crypto_signature_api_t
- continuous_mode : cac_cfg_t
- contrast : display_correction_t , display_output_cfg_t
- contrastDecrease : slcdc_api_t
- contrastIncrease : slcdc_api_t
- control : can_api_t , sdmmc_api_t
- control_point_val : sf_ble_prf_value_t
- control_point_value : sf_ble_anp_ancp_change_t , sf_ble_hrp_cp_change_t , sf_ble_tip_cp_change_t
- conv_end_irq : sdadc_instance_ctrl_t
- conversion_end_ipl : adc_on_sdadc_cfg_t
- coordinate : display_layer_t
- correction : display_api_t
- correction_proc_order : glcd_cfg_t
- count_direction : timer_info_t
- count_edge : agt_input_capture_extend_t
- count_formula : sdadc_channel_cfg_t
- count_source : agt_input_capture_extend_t , timer_on_agt_cfg_t
- counter : input_capture_callback_args_t , input_capture_capture_t , sf_el_fx_media_global_open_info_t
- counterGet : timer_api_t , wdt_api_t
- countIncrement : sf_thread_monitor_api_t
- country_code : sf_cellular_network_status_t
- CRA : dtc_reg_t
- CRA_b : dtc_reg_t
- CRAH : dtc_reg_t
- CRAL : dtc_reg_t
- CRB : dtc_reg_t
- crcResultGet : crc_api_t
- createKey : aes_api_t
- crossing_detected : lvd_status_t
- crypto_ctrl : tdes_ctrl_t
- csd_version : sdmmc_info_t
- csrk_key : sf_ble_sec_info_t
- ctsrts_en : uart_cfg_t
- ctsu_clock : ctsu_correction_info_t
- ctsuchac0 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac1 : ctsu_cfg_t , ctsu_instance_ctrl_t

- ctsuchac2 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac3 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac4 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac_register_count : bsp_feature_ctsu_t
- ctsuchtrc0 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc1 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc2 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc3 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc4 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc_register_count : bsp_feature_ctsu_t
- ctsucr0_mask : bsp_feature_ctsu_t
- ctsucr1 : ctsu_instance_ctrl_t
- ctsucr1_mask : bsp_feature_ctsu_t
- ctsumch0_mask : bsp_feature_ctsu_t
- ctsumch1_mask : bsp_feature_ctsu_t
- ctsuso0 : ctsu_ctsuwr_t
- ctsuso1 : ctsu_ctsuwr_t
- ctsussc : ctsu_ctsuwr_t
- ctsuwr : ctsu_correction_info_t
- curr_cmd_port : sf_wifi_qca4010_instance_cfg_t
- curr_data_port : sf_wifi_qca4010_instance_cfg_t
- curr_socket_index : sf_wifi_qca4010_instance_cfg_t
- currbuf : trng_ctrl_t
- current_buffer_index : sf_audio_record_i2s_instance_ctrl_t
- current_count : sf_thread_monitor_thread_counter_t
- current_sample_count : sf_adc_periodic_instance_ctrl_t
- current_slave : rspi_instance_ctrl_t
- current_state : lvd_status_t , sf_ble_prf_rtus_time_updt_state_t
- current_time : sf_ble_cts_curr_time_ntf_t , sf_ble_prf_tip_write_data_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- d -

- dac_mode : dac8_extended_cfg_t
- DAR : dtc_reg_t
- data : can_frame_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_INFO_DATA , sf_ble_adv_data_t , sf_ble_attr_info_t , sf_ble_scan_t , uart_callback_args_t
- data_bits : uart_cfg_t
- data_buffer_index : sf_adc_periodic_instance_ctrl_t
- data_buffer_length : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- data_bytes : sci_uart_instance_ctrl_t
- data_enable_polarity : display_output_cfg_t
- data_flash : flash_info_t
- data_flash_bgo : flash_cfg_t
- data_format : dac8_instance_ctrl_t , dac_cfg_t
- data_len : sf_ble_char_multiple_read_rsp_t , sf_ble_char_read_by_handle_rsp_t ,

sf_ble_char_read_by_uuid_rsp_t , sf_ble_scan_t
- data_length : key_installation_key_t , r_crypto_data_handle_t , sf_ble_char_write_req_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t , sf_crypto_data_handle_t
- data_length_code : can_frame_t
- data_lines : qspi_instance_ctrl_t
- data_size : sf_audio_record_i2s_instance_ctrl_t
- data_state : sf_cellular_socket_info_t
- data_type : sf_audio_playback_common_instance_ctrl_t
- dataGet : ctsu_api_t , sf_touch_ctsu_api_t
- dataTypeGet : sf_audio_playback_hw_api_t
- dave2d_buffer_cache_enabled : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- day : sf_ble_prf_cts_date_time_t
- day_of_week : sf_ble_prf_cts_curr_time_t
- dayofweek_match : rtc_alarm_time_t
- days_since_update : sf_ble_cts_ref_time_t
- decrypt : aes_api_t , rsa_api_t , tdes_api_t
- decryptCrt : rsa_api_t
- deep_standby_cancel_edge : lpmv2_mcu_cfg_t
- deep_standby_cancel_source : lpmv2_mcu_cfg_t
- delay : ptp_cfg_t , ptp_instance_ctrl_t
- desc_handle : sf_ble_char_desc_discovery_rsp_t
- desc_hdl : RBLE_GATT_CHAR_DESC_LIST
- desired_char : RBLE_GATT_DISC_CHAR_REQ
- desired_svc : RBLE_GATT_DISC_SVC_REQ
- dest_addr_mode : transfer_info_t
- detection_response : lvd_cfg_t
- dev_state : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- device : ptp_cfg_t , ptp_instance_ctrl_t , sf_el_gx_callback_args_t
- device_count : sf_i2c_bus_t , sf_spi_bus_t
- device_count_mutex : sf_i2c_bus_t , sf_spi_bus_t
- device_type : sdmmc_info_t , sf_ble_prf_hid_report_desc_t
- dhcpServerStart : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- dhcpServerStop : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- diagnosis : ctsu_api_t
- digfilter : cac_ref_clock_config_t
- direct_addr_type : sf_ble_adv_info_t
- direct_bd_addr : sf_ble_adv_info_t
- disable : elc_api_t , external_irq_api_t , input_capture_api_t , keymatrix_api_t , transfer_api_t
- disableINFABTnotification : ptp_api_t
- disableTimer : ptp_api_t
- disc_mode : sf_ble_adv_info_t
- disc_time : sf_ble_cfg_t
- disconnect : sf_ble_api_t
- discovery_type : sf_ble_char_discovery_req_t , sf_ble_scan_info_t , sf_ble_service_discovery_req_t
- DISEL : dtc_reg_t
- disp_en : sf_ble_sm_tk_info_t
- display_cyc : display_timing_t
- display_list_flushed : sf_el_gx_instance_ctrl_t
- dithering_mode : glcd_cfg_t
- dithering_on : display_output_cfg_t
- dithering_pattern_A : glcd_cfg_t
- dithering_pattern_B : glcd_cfg_t
- dithering_pattern_C : glcd_cfg_t

- dithering_pattern_D : glcd_cfg_t
- divider : cac_meas_clock_config_t , cac_ref_clock_config_t , cgc_clock_cfg_t
- DM : dtc_reg_t
- dma_req_ipl : sdmmc_cfg_t
- do_run : sf_cellular_comms_extend_cfg_t
- dodir : doc_data_t
- dodsr : doc_data_t
- domain_params : sf_crypto_key_cfg_t , sf_crypto_key_instance_ctrl_t
- dri_marker : jpeg_encode_cfg_t
- drift_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- drive_volt_gen : slcdc_cfg_t
- driver_packets_queued : NX_REC
- driver_rx_bd : NX_REC
- driver_rx_bd_index : NX_REC
- driver_task_priority : sf_wifi_on_gt202_cfg_t
- driver_tx_bd : NX_REC
- driver_tx_bd_in_use : NX_REC
- driver_tx_bd_index : NX_REC
- driver_tx_packet_queue : NX_REC
- driver_tx_packet_queue_end : NX_REC
- driver_tx_release_index : NX_REC
- dst_offset : sf_ble_cts_local_time_t , sf_ble_prf_ndcs_time_dst_t
- dtc_state_in_snooze : lpmv2_mcu_cfg_t
- dtc_transfer_length : sf_adc_periodic_instance_ctrl_t
- dtim : sf_wifi_cfg_t
- DTS : dtc_reg_t
- dummy_read_completed : riic_instance_ctrl_t
- duplicate_filt : sf_ble_scan_info_t
- duty_cycle : timer_cfg_t
- duty_cycle_unit : timer_cfg_t
- dutyCycleSet : timer_api_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- e -

- each_len : RBLE_GATT_INFO_DATA
- ebr : sf_el_fx_media_boot_record_table_info_t
- EcdhSharedSecretCompute : sf_crypto_key_api_t
- echo : sf_console_cfg_t , sf_console_instance_ctrl_t
- edge : cac_ref_clock_config_t , input_capture_cfg_t
- ediv : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- edmac_ptr : NX_REC
- elc_event : adc_info_t , timer_info_t
- elc_peripheral : adc_info_t
- elem_index : sf_touch_ctsu_button_cfg_t
- elmt : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t
- elmt_hdl : RBLE_GATT_RELIABLE_WRITE
- enable : display_brightness_t , display_contrast_t , elc_api_t , external_irq_api_t , gamma_correction_t , input_capture_api_t , keymatrix_api_t , transfer_api_t

- enable_charge_pump : dac8_extended_cfg_t , dac_extended_cfg_t
- enable_filter : gpt_input_capture_extend_t
- enable_level : gpt_input_capture_extend_t
- enableINFABTnotification : ptp_api_t
- encoded_lines : jpeg_encode_instance_ctrl_t
- encrypt : aes_api_t , rsa_api_t , tdes_api_t
- encryptFinal : aes_api_t
- encryption : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_scan_t
- end : sf_message_instance_range_t , sf_audio_playback_instance_ctrl_t
- end_handle : sf_ble_char_discovery_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- end_hdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_SET_PERM , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , RBLE_GATT_SVC_RANGE_LIST
- end_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- endian : display_output_cfg_t , pdc_cfg_t , pdc_instance_ctrl_t
- endian_flag : crypto_cfg_t
- energy_expended : sf_ble_hrp_api_hrmeas_t
- entry_len : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t
- eptpc_flag : ptp_instance_ctrl_t
- erase : flash_api_t , qspi_api_t , sdmmc_api_t , sf_memory_api_t
- erase_block_count : sf_el_lx_nor_callback_args_t
- erase_block_number : sf_el_lx_nor_callback_args_t
- erase_sector_count : sdmmc_info_t
- eri_ipl : i2c_cfg_t , spi_cfg_t , uart_cfg_t
- eri_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t
- err : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- err_irq : flash_hp_instance_ctrl_t
- err_irq_ipl : flash_cfg_t
- error : sf_crypto_callback_args_t , sf_crypto_hash_callback_args_t , sf_el_gx_callback_args_t
- error_adjustment_type : rtc_cfg_t
- error_adjustment_value : rtc_cfg_t
- error_check_index : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t
- error_code : jpeg_decode_instance_ctrl_t
- error_ipl : can_cfg_t
- error_irq : can_instance_ctrl_t
- errorAdjustmentModeSet : rtc_api_t
- errorAdjustmentSet : rtc_api_t
- ether_frame_type : ptpedmac_callback_args_t
- etherc_ptr : NX_REC
- event : adc_callback_args_t , cac_callback_args_t , can_callback_args_t , cgc_callback_args_t , ctsu_callback_args_t , display_callback_args_t , doc_callback_args_t , doc_cfg_t , doc_instance_ctrl_t , elc_link_t , flash_callback_args_t , i2c_callback_args_t , i2s_callback_args_t , input_capture_callback_args_t , pdc_callback_args_t , ptp_callback_args_t , ptpedmac_callback_args_t , rtc_callback_args_t , sdmmc_callback_args_t , sf_adc_periodic_callback_args_t , sf_audio_playback_hw_callback_args_t , sf_ble_event_info_t , sf_cellular_callback_args_t , sf_comms_callback_args_t , sf_crypto_callback_args_t , sf_el_gx_callback_args_t , sf_el_lx_nor_callback_args_t , sf_external_irq_cfg_t , sf_message_callback_args_t , sf_power_profiles_v2_callback_args_t , sf_wifi_callback_args_t , spi_callback_args_t , timer_callback_args_t , uart_callback_args_t
- event_class : sf_message_subscriber_list_t
- event_type : sf_ble_scan_t , sf_touch_panel_v2_payload_t

- eventflag : sf_block_media_sdmmc_instance_ctrl_t , sf_uart_comms_instance_ctrl_t
- eventInfoGet : fmi_api_t
- events : sf_jpeg_decode_instance_ctrl_t
- exe_wr_ena : RBLE_GATT_EXE_WR_CHAR_REQ
- expect_resp_size : RBLE_GATT_UUID_TYPE
- expected_result_size : sf_ble_char_multiple_read_req_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- f -

- fade_control : display_layer_t
- fade_speed : display_layer_t
- fade_status : display_status_t
- fclk_div : cgc_system_clock_cfg_t
- ferr_interrupt_enabled : cac_cfg_t
- fifo_access_bytes : ssi_instance_ctrl_t
- fifo_addr : UX_HCD_SYNERGY_FIFO
- fifo_ctrl : UX_HCD_SYNERGY_FIFO
- fifo_depth : sci_uart_instance_ctrl_t
- fifo_mode : crc_cfg_t , crc_instance_ctrl_t
- fifo_num_stages : bsp_feature_ssi_t
- fifo_sel : UX_HCD_SYNERGY_FIFO
- filt_policy : sf_ble_scan_info_t
- filter : comparator_cfg_t
- filter_enable : external_irq_cfg_t
- filterDisable : external_irq_api_t
- filterEnable : external_irq_api_t
- first_coefficient : ctsu_correction_info_t
- first_val : ctsu_correction_info_t
- flag_stable_meas : sf_ble_blp_meas_info_t , sf_ble_prf_htp_temp_info_t
- flags : agt_input_capture_instance_ctrl_t , sf_audio_playback_common_instance_ctrl_t , sf_ble_blp_meas_info_t , sf_ble_hrp_api_hrmeas_t , sf_ble_prf_htp_temp_info_t , sf_touch_panel_v2_instance_ctrl_t
- flash_cf_macros : bsp_feature_flash_lp
- flash_clock_src : bsp_feature_flash_lp
- flush : sf_memory_api_t
- format : display_input_cfg_t , display_output_cfg_t
- format_status : sf_el_fx_media_partition_data_info_t
- fotaCheck : sf_cellular_api_t
- fotaStart : sf_cellular_api_t
- fotaStop : sf_cellular_api_t
- fractions256 : sf_ble_prf_cts_curr_time_t
- fragmentation : sf_wifi_cfg_t
- frame_end_ipl : pdc_cfg_t
- frame_end_irq : pdc_instance_ctrl_t
- frame_format : ptp_cfg_t , ptp_instance_ctrl_t
- frame_type : can_mailbox_t
- freq_hz_min : sf_spi_bus_t
- frequency_error_ipl : cac_cfg_t

- frequency_error_irq : cac_instance_ctrl_t
- fw_version : sf_cellular_info_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- g -

- g : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- gain : gamma_correction_t
- gap_name : sf_ble_provisioning_t
- gap_role : sf_ble_provisioning_t
- gateway : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- gattAddCustomProfiles : sf_ble_api_t
- gattCharDescDiscovery : sf_ble_api_t
- gattCharDiscovery : sf_ble_api_t
- gattCharExecuteWrite : sf_ble_api_t
- gattCharRead : sf_ble_api_t
- gattCharWrite : sf_ble_api_t
- gattCharWriteLocal : sf_ble_api_t
- gattSendIndicate : sf_ble_api_t
- gattSendNotify : sf_ble_api_t
- gattServiceDiscovery : sf_ble_api_t
- gattWriteResponse : sf_ble_api_t
- generator_point : sf_crypto_key_cfg_t , sf_crypto_key_instance_ctrl_t
- getGcmTag : aes_api_t
- getLocalClock : ptp_api_t
- getMasterPortID : ptp_api_t
- getMessageReceptionConfig : ptp_api_t
- getSyncInfo : ptp_api_t
- getWorst10Values : ptp_api_t
- global_open : sf_el_fx_media_info_t
- group_b_sensors_allowed : bsp_feature_adc_t
- gtioca : timer_on_gpt_cfg_t
- gtioca_output_enabled : gpt_instance_ctrl_t
- gtiocb : timer_on_gpt_cfg_t
- gtiocb_output_enabled : gpt_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- h -

- handle : sf_ble_char_read_by_uuid_rsp_t , sf_ble_char_read_req_t , sf_ble_char_write_req_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t
- handles : sf_ble_char_multiple_read_req_t
- handles_cnt : sf_ble_char_read_req_t

- has_bclk : bsp_feature_cgc_t
- has_card_detection : bsp_feature_sdhi_t
- has_chargepump : bsp_feature_dac_t
- has_davrefcr : bsp_feature_dac_t
- has_digital_filter : bsp_feature_lvd_t
- has_dssby : bsp_feature_lpmv2_t
- has_ethernet : bsp_feature_ioport_t
- has_fclk : bsp_feature_cgc_t
- has_ir_flag : bsp_feature_icu_t
- has_lcd_clock : bsp_feature_cgc_t
- has_pclka : bsp_feature_cgc_t
- has_pclkb : bsp_feature_cgc_t
- has_pclkc : bsp_feature_cgc_t
- has_pclkd : bsp_feature_cgc_t
- has_sample_hold_reg : bsp_feature_adc_t
- has_sdadc_clock : bsp_feature_cgc_t
- has_sdram_clock : bsp_feature_cgc_t
- has_subosc_speed : bsp_feature_cgc_t
- has_usb_clock_div : bsp_feature_cgc_t
- has_vbatt_pins : bsp_feature_ioport_t
- hash : hash_ctrl_t
- hash_context : sf_crypto_hash_instance_ctrl_t
- hash_type : sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t
- hashFinal : sf_crypto_hash_api_t
- hashInit : sf_crypto_hash_api_t
- hashSign : dsa_api_t
- hashUpdate : hash_api_t , sf_crypto_hash_api_t
- hashVerify : dsa_api_t
- hc : sdmmc_info_t
- header : sf_audio_playback_data_t
- heart_rate_measure : sf_ble_hrp_api_hrmeas_t
- help : sf_console_command_t
- high_speed_freq_hz : bsp_feature_cgc_t
- high_throughput : sf_wifi_cfg_t
- hoco_freq_hz : bsp_feature_cgc_t
- hoco_state : cgc_clocks_cfg_t
- horizontal_resolution : jpeg_encode_cfg_t , jpeg_encode_raw_image_parameters
- horizontal_stride : jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , jpeg_encode_raw_image_parameters
- horizontal_subsample : jpeg_decode_instance_ctrl_t
- horizontalStrideSet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- hour : sf_ble_prf_cts_date_time_t
- hour_match : rtc_alarm_time_t
- hours_since_update : sf_ble_cts_ref_time_t
- hs_timing : sdmmc_info_t
- hsize : display_input_cfg_t , glcd_ctrl_t
- hsize_pixels : sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- hstride : display_input_cfg_t
- hsync : pdc_state_t
- hsync_polarity : pdc_cfg_t , pdc_instance_ctrl_t
- htiming : display_output_cfg_t
- hw : sdmmc_cfg_t , sdmmc_instance_ctrl_t
- hw_cfg : rtc_cfg_t

- hw_mode : sf_wifi_cfg_t , sf_wifi_qca4010_cfg_t , sf_wifi_scan_t
- hysteresis : sf_touch_ctsu_button_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- i -

- i2c_hw_err_event : i2c_callback_args_t
- iclk_div : bsp_feature_cgc_t , cgc_system_clock_cfg_t
- id : can_frame_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , sf_ble_blp_meas_info_t
- id_key : sf_ble_sec_info_t
- id_mode : can_cfg_t , can_instance_ctrl_t
- idCodeSet : flash_api_t
- idle_err_ipl : i2s_cfg_t
- ikey_dist : sf_ble_bonding_req_ind_t , sf_ble_bonding_start_t
- ikeys : sf_ble_bonding_response_t
- image_size : jpeg_encode_callback_args_t
- imageParameterSet : jpeg_encode_api_t
- imageSizeGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- imageSubsampleSet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- imei : sf_cellular_info_t
- imsi : sf_cellular_network_status_t
- in_progress : transfer_properties_t
- in_use : sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- incl : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t::incl_list_u
- index : sf_ble_bonding_req_ind_t , sf_wifi_qca4010_provisioning_t , sf_audio_playback_instance_ctrl_t
- indicateEvent : ptp_api_t
- infabt_flag : ptp_instance_ctrl_t
- info : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , slcdc_instance_ctrl_t
- info_transfer : pdc_instance_ctrl_t
- infoGet : adc_api_t , can_api_t , comparator_api_t , dac_api_t , flash_api_t , i2s_api_t , input_capture_api_t , opamp_api_t , qspi_api_t , rtc_api_t , sdmmc_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_cellular_api_t , sf_memory_api_t , sf_wifi_api_t , timer_api_t , transfer_api_t , uart_api_t
- inherit_frame_layer : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- init : analog_connect_api_t , cgc_api_t , elc_api_t , fmi_api_t , ioport_api_t , lpmv2_api_t
- init_done : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- init_filt_type : sf_ble_connection_t
- init_status : sf_el_fx_media_info_t
- input : display_cfg_t , display_runtime_cfg_t , opamp_trim_args_t , sdadc_channel_cfg_t , sf_console_instance_ctrl_t
- input_data_format : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- inputBufferSet : jpeg_decode_api_t , jpeg_encode_api_t , sf_jpeg_decode_api_t
- inputRegisterWrite : doc_api_t
- inst_idx : sf_ble_onboard_profile_cccd_changed_t , sf_ble_prf_hid_change_event_t , sf_ble_prf_hid_report_ind_t
- instance_range : sf_message_subscriber_t

- int_irq : ssi_instance_ctrl_t
- interfaceGet : crypto_api_t
- intv : st_sf_ble_prf_htp_meas_intv_val_t
- invert : comparator_cfg_t , sdadc_channel_cfg_t
- io_cap : sf_ble_bonding_req_ind_t , sf_ble_bonding_response_t
- io_port_state : lpmv2_mcu_cfg_t
- iocap : sf_ble_bonding_start_t
- ioctl : sf_block_media_api_t
- IoIntEnable : sdmmc_api_t
- ip_addr : sf_cellular_info_t , sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- ip_addr_mode : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- ip_ptr : NX_REC
- ipAddressCfg : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- ir_flag_stat : dmac_instance_ctrl_t
- irq : agt_instance_ctrl_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , gpt_instance_ctrl_t , icu_instance_ctrl_t , kint_instance_ctrl_t , NX_REC , pdc_instance_ctrl_t , transfer_info_t
- irq_ipl : comparator_cfg_t , doc_cfg_t , external_irq_cfg_t , flash_cfg_t , keymatrix_cfg_t , pdc_cfg_t , ptp_cfg_t , ptpedmac_cfg_t , timer_cfg_t , transfer_cfg_t
- irqDisable : rtc_api_t
- irqEnable : rtc_api_t
- is_dac_ramped_up : sf_audio_playback_hw_dac_instance_ctrl_t
- is_data_mode_on : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- is_opened : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- is_signed : sf_audio_playback_data_type_t
- iwdt_open : iwdt_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- j -

- jdti_ipl : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- jedi_ipl : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- jpegbuffer_size : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- k -

- key : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t
- key_code : sf_ble_sm_key_ind_t
- key_data_length : sf_crypto_signature_context_t
- key_format : key_installation_key_t , rsa_key_t
- key_size : key_installation_key_t , sf_ble_bonding_start_t , sf_ble_sec_enc_start_ind_t ,

sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_cfg_t , sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- key_type : sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_cfg_t , sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- keyCreate : ecc_api_t , rsa_api_t
- keyGenerate : sf_crypto_key_api_t
- keyInstall : key_installation_api_t , sf_crypto_key_installation_api_t
- keySet : arc4_api_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- l -

- last_payload : sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- lastCaptureGet : input_capture_api_t
- layer : display_cfg_t , display_runtime_cfg_t
- layerChange : display_api_t
- lcdClockCfg : cgc_api_t
- lcdClockDisable : cgc_api_t
- lcdClockEnable : cgc_api_t
- le_scan_interval : sf_ble_prf_scps_scan_intv_t
- le_scan_window : sf_ble_prf_scps_scan_intv_t
- led_count : bsp_leds_t
- len : RBLE_GATT_INFO_DATA , RBLE_GATT_QUERY_RESULT , sf_ble_attr_info_t
- length : adc_info_t , arc4_cfg_t , hash_ctrl_t , rsa_key_t , sf_adc_periodic_instance_ctrl_t , sf_ble_gatt_attr_event_t , sf_cellular_callback_args_t , sf_wifi_callback_args_t , transfer_info_t
- line_descending_enable : display_input_cfg_t
- line_detect_ipl : display_cfg_t
- lines_repeat_enable : display_input_cfg_t
- lines_repeat_times : display_input_cfg_t
- lines_to_encoded : jpeg_encode_instance_ctrl_t
- linesDecodedGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- link_count : elc_cfg_t
- link_established : NX_REC
- link_list : elc_cfg_t
- link_quality : sf_wifi_info_t
- linkBreak : elc_api_t
- linkCheck : ptpedmac_api_t
- linkProcess : ptpedmac_api_t
- linkSet : elc_api_t
- list : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t ,

RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t::incl_list_u

- list_128 : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t
- listen : sf_ble_api_t
- loaded : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- local_advertise : phy_record_t
- local_ip : sf_cellular_nsal_cfg_t , sf_cellular_socket_info_t
- local_port : sf_cellular_socket_info_t
- local_time : sf_ble_prf_tip_write_data_t
- lock : bsp_lock_t , sf_comms_api_t , sf_crypto_api_t , sf_i2c_api_t , sf_spi_api_t
- locked : sf_spi_instance_ctrl_t
- lockWait : sf_i2c_api_t , sf_spi_api_t
- loco_state : cgc_clocks_cfg_t
- loop_timeout : sf_audio_playback_data_t
- loopback : spi_on_rspi_cfg_t
- low_power_mode : lpmv2_cfg_t
- low_speed_max_freq_hz : bsp_feature_cgc_t
- low_speed_pclk_div_min : bsp_feature_cgc_t
- low_voltage_max_freq_hz : bsp_feature_cgc_t
- low_voltage_pclk_div_min : bsp_feature_cgc_t
- lower_level : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- lower_lvl_cfg : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- lowPowerApply : sf_power_profiles_v2_api_t
- lowPowerCfg : lpmv2_api_t
- lowPowerModeEnter : lpmv2_api_t
- ltk_key : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- lvd_callback_args : lvd_instance_ctrl_t
- lvl : sb_ble_prf_ias_set_alert_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- m -

- mac_addr : sf_wifi_callback_args_t , sf_wifi_cfg_t , sf_wifi_qca4010_status_t
- macAddressGet : sf_wifi_api_t
- macAddressSet : sf_wifi_api_t
- mailbox : can_callback_args_t
- mailbox_count : can_cfg_t , can_instance_ctrl_t
- mailbox_id : can_mailbox_t
- mailbox_rx_ipl : can_cfg_t
- mailbox_rx_irq : can_instance_ctrl_t
- mailbox_tx_ipl : can_cfg_t
- mailbox_tx_irq : can_instance_ctrl_t
- mailbox_type : can_mailbox_t
- main_osc_freq_hz : bsp_feature_cgc_t
- mainclock_drive : bsp_feature_cgc_t

- mainosc_state : cgc_clocks_cfg_t
- major : ssp_pack_version_t
- manufacturer_id : qspi_instance_ctrl_t
- masterReadSlaveWrite : i2c_api_slave_t
- masterWriteSlaveRead : i2c_api_slave_t
- max_backoffs : sf_cellular_qctlcatm1_socket_cfg_t
- max_clock_frequency : bsp_feature_sdhi_t
- max_clock_rate : sdmmc_info_t
- max_enc_size : sf_ble_bonding_req_ind_t
- max_eraseable_size : qspi_instance_ctrl_t
- max_key_size : sf_ble_bonding_response_t
- max_resolution : adc_instance_ctrl_t
- max_resp_length : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- max_rto : sf_cellular_qctlcatm1_socket_cfg_t
- max_slaves : sf_ble_cfg_t
- max_stations : sf_wifi_cfg_t
- maximum_count : sf_thread_monitor_counter_min_max_t , sf_thread_monitor_thread_counter_t
- mbr : sf_el_fx_media_boot_record_table_info_t
- mclock_only : bsp_feature_can_t
- md : ctsu_cfg_t , ctsu_instance_ctrl_t
- MD : dtc_reg_t
- mday_match : rtc_alarm_time_t
- meas_info : sf_ble_blp_meas_recv_data_t
- meas_sts : sf_ble_blp_meas_info_t
- measurement_clock : cac_instance_ctrl_t
- measurement_end_ipl : cac_cfg_t
- measurement_end_irq : cac_instance_ctrl_t
- measurements_info : sf_ble_hrp_api_meas_ntf_t
- media_info : sf_el_fx_instance_ctrl_t
- media_type : sdmmc_hw_t
- mei_interrupt_enabled : cac_cfg_t
- memory_capacity : qspi_instance_ctrl_t
- memory_end_address : sf_memory_region_info_t
- memory_free_sectors : sf_el_fx_media_info_t
- memory_pool_size : sf_crypto_cfg_t
- memory_start_address : sf_memory_region_info_t
- memory_total_sectors : sf_el_fx_media_info_t
- memory_type : qspi_instance_ctrl_t
- menu_name : sf_console_menu_t
- menu_prev : sf_console_menu_t
- message_bytes : sf_crypto_hash_context_t
- message_bytes_buffered : sf_crypto_hash_context_t
- message_format : sf_crypto_signature_rsa_specific_params_t
- message_mode : can_cfg_t , can_instance_ctrl_t
- mfg_name : sf_cellular_info_t
- middle_speed_max_freq_hz : bsp_feature_cgc_t
- min : sf_ble_prf_cts_date_time_t
- min_match : rtc_alarm_time_t
- min_program_size_bytes : qspi_info_t
- min_stabilization_wait_us : comparator_info_t , opamp_info_t
- min_wait_time_hs_us : bsp_feature_opamp_t
- min_wait_time_lp_us : bsp_feature_opamp_t
- min_wait_time_ms_us : bsp_feature_opamp_t
- min_wait_time_us : bsp_feature_acmphs_t

- minimum_count : sf_thread_monitor_counter_min_max_t , sf_thread_monitor_thread_counter_t
- minimum_erase_size : sf_memory_region_info_t
- minimum_write_size : sf_memory_region_info_t
- minor : ssp_pack_version_t
- mint_irq : ptp_instance_ctrl_t
- moco_state : cgc_clocks_cfg_t
- mode : adc_cfg_t , adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , comparator_cfg_t , gpt_input_capture_instance_ctrl_t , input_capture_cfg_t , opamp_on_opamp_cfg_t , sdadc_calibrate_args_t , sdadc_instance_ctrl_t , sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_status_t , timer_cfg_t , transfer_info_t
- mode_fault : spi_cfg_t
- modify : slcdc_api_t
- modrv_mask : bsp_feature_cgc_t
- modrv_shift : bsp_feature_cgc_t
- mon_match : rtc_alarm_time_t
- monitor_1_hi_threshold : bsp_feature_lvd_t
- monitor_1_low_threshold : bsp_feature_lvd_t
- monitor_2_hi_threshold : bsp_feature_lvd_t
- monitor_2_low_threshold : bsp_feature_lvd_t
- monitor_ipl : lvd_cfg_t
- monitor_number : lvd_callback_args_t , lvd_cfg_t , lvd_instance_ctrl_t
- month : sf_ble_prf_cts_date_time_t
- mosi_idle : spi_on_rspi_cfg_t
- MRA : dtc_reg_t
- MRA_b : dtc_reg_t
- MRB : dtc_reg_t
- MRB_b : dtc_reg_t
- msgbuf : hash_ctrl_t
- multicastListAdd : sf_wifi_api_t
- multicastListDelete : sf_wifi_api_t
- multiple_partitions_status : sf_el_fx_media_partition_info_t
- multiplier : cgc_clock_cfg_t
- mute : i2s_api_t
- mutex : sf_adc_periodic_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_i2s_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_jpeg_decode_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t , sf_uart_comms_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- n -

- nak_response : sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- nattempts : trng_cfg_t , trng_ctrl_t
- nb_entry : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t ,

RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t

- nb_resp : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t
- nb_uuid : RBLE_GATT_READ_CHAR_REQ
- nb_writes : RBLE_GATT_WRITE_RELIABLE_REQ
- nbiot_band_selection : sf_cellular_qctlcatm1_extended_cfg_t
- negation_delay : lvd_extend_t
- negation_delay_clock : bsp_feature_lvd_t
- netmask : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- networkConnect : sf_cellular_api_t
- networkDisconnect : sf_cellular_api_t
- networkStatusGet : sf_cellular_api_t
- new_alert : sf_ble_anp_api_new_alert_ntf_t
- new_conn_fd : sf_cellular_socket_info_t
- new_line : sf_console_instance_ctrl_t
- next_dst : sf_ble_prf_tip_write_data_t
- next_length : sf_audio_playback_common_instance_ctrl_t
- noise_level : sf_wifi_info_t
- noisecancel_en : uart_on_sci_cfg_t
- nor_driver_initialize : sf_block_media_on_lx_nor_cfg_t
- notify_request : riic_slave_instance_ctrl_t
- num_address_bytes : qspi_instance_ctrl_t
- num_blocks : transfer_info_t
- num_buttons : sf_touch_ctsu_cfg_t
- num_commands : sf_console_menu_t
- num_elements : ctsu_instance_ctrl_t , sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- num_erase_sizes : qspi_info_t
- num_moving_average : ctsu_cfg_t , ctsu_instance_ctrl_t
- num_new_samples : sf_adc_periodic_callback_args_t
- num_pref_ops : sf_cellular_cfg_t
- num_regions : flash_fmi_regions_t
- num_rx : ctsu_cfg_t
- num_sliders : sf_touch_ctsu_cfg_t
- num_states : adc_sample_state_t
- num_tx : ctsu_cfg_t
- num_uarts : sf_wifi_qca4010_cfg_t , sf_wifi_qca4010_instance_cfg_t
- num_wheels : sf_touch_ctsu_cfg_t
- number : sf_touch_ctsu_cfg_t
- number_of_buffers : sf_message_instance_ctrl_t
- number_of_connections : analog_connect_table_t
- number_of_nodes : sf_message_subscriber_list_t
- number_of_pins : ioport_cfg_t
- number_of_regions : sf_memory_info_t
- number_of_subscriber_groups : sf_message_instance_ctrl_t
- nwscanseq : sf_cellular_qctlcatm1_extended_cfg_t
- nx_driver_phy_polling_requested : NX_REC
- nx_state : NX_REC

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- o -

- off_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- offset : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_READ_CHAR_REQ , sf_ble_char_read_req_t , sf_ble_char_write_req_t , sf_ble_gatt_attr_event_t , sf_ble_write_cmd_event_data_t , sf_el_fx_media_partition_data_info_t
- offset_byte : dmac_instance_ctrl_t , transfer_on_dmac_cfg_t
- offsetSet : adc_api_t
- ok_check_index : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t
- on_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- onbpClientReadChar : sf_ble_onboard_profile_api_t
- onbpClientWriteCCCD : sf_ble_onboard_profile_api_t
- onbpClientWriteChar : sf_ble_onboard_profile_api_t
- onbpDisable : sf_ble_onboard_profile_api_t
- onbpEnable : sf_ble_onboard_profile_api_t
- onbpServerSendIndication : sf_ble_onboard_profile_api_t
- onbpServerSendNotification : sf_ble_onboard_profile_api_t
- onbpServerWriteData : sf_ble_onboard_profile_api_t
- one_shot : gpt_instance_ctrl_t
- oob_data_flg : sf_ble_bonding_req_ind_t
- op : sf_cellular_cfg_t
- op_name : sf_cellular_network_status_t , sf_cellular_op_t
- op_name_format : sf_cellular_op_t
- op_select_mode : sf_cellular_cfg_t
- open : adc_api_t , aes_api_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , arc4_api_t , arc4_ctrl_t , cac_api_t , can_api_t , can_instance_ctrl_t , comparator_api_t , crc_api_t , crc_instance_ctrl_t , crypto_api_t , ctsu_api_t , ctsu_instance_ctrl_t , dac_api_t , display_api_t , doc_api_t , doc_instance_ctrl_t , dsa_api_t , ecc_api_t , external_irq_api_t , flash_api_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , icu_instance_ctrl_t , input_capture_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , kint_instance_ctrl_t , lvd_api_t , opamp_api_t , pdc_api_t , pdc_instance_ctrl_t , ptp_api_t , ptp_instance_ctrl_t , ptpedmac_api_t , ptpedmac_instance_ctrl_t , qspi_api_t , qspi_instance_ctrl_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rsa_api_t , rtc_api_t , rtc_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_uart_instance_ctrl_t , sdmmc_api_t , sdmmc_instance_ctrl_t , sf_adc_periodic_api_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_api_t , sf_audio_playback_common_instance_ctrl_t , sf_audio_playback_hw_api_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_api_t , sf_audio_record_i2s_instance_ctrl_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_block_media_ram_instance_ctrl_t , sf_block_media_sdmmc_instance_ctrl_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t , sf_el_fx_instance_ctrl_t , sf_el_gx_api_t , sf_el_lx_nor_instance_ctrl_t , sf_external_irq_api_t , sf_external_irq_instance_ctrl_t , sf_i2c_api_t , sf_jpeg_decode_api_t , sf_jpeg_decode_instance_ctrl_t , sf_memory_api_t , sf_memory_qspi_nor_instance_ctrl_t , sf_message_api_t , sf_power_profiles_v2_api_t ,

sf_power_profiles_v2_ctrl_t , sf_socket_api_t , sf_spi_api_t , sf_thread_monitor_api_t , sf_thread_monitor_instance_ctrl_t , sf_touch_ctsu_api_t , sf_touch_ctsu_instance_ctrl_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_touch_panel_v2_instance_ctrl_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , ssi_instance_ctrl_t , sf_audio_playback_instance_ctrl_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t , wdt_api_t

- open_counter : sf_crypto_instance_ctrl_t
- open_status : sf_el_fx_media_partition_data_info_t
- opened : adc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , lvd_instance_ctrl_t , opamp_instance_ctrl_t , sdadc_instance_ctrl_t
- operating_channel_mask : opamp_status_t
- operating_mode : i2s_cfg_t , spi_cfg_t
- operation_context : sf_crypto_signature_instance_ctrl_t
- operation_mode : can_instance_ctrl_t , sf_crypto_signature_context_t
- operation_state : sf_crypto_signature_instance_ctrl_t
- operator_code : sf_cellular_network_status_t
- oscStopDetect : cgc_api_t
- oscStopStatusClear : cgc_api_t
- outbuffer_size : jpeg_decode_instance_ctrl_t
- output : display_cfg_t
- output_amplifier_enabled : dac_cfg_t
- output_buffer_size : jpeg_encode_instance_ctrl_t
- output_data_format : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- output_enabled : gpt_output_pin_t
- output_inverted : timer_on_agt_cfg_t
- output_port_enable : lpmv2_mcu_cfg_t
- outputBufferSet : jpeg_decode_api_t , jpeg_encode_api_t , sf_jpeg_decode_api_t
- outputEnable : comparator_api_t
- over_current : adc_cfg_t , adc_instance_ctrl_t
- overflow_ipl : cac_cfg_t
- overflow_irq : agt_input_capture_instance_ctrl_t , cac_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- overflow_irq_ipl : input_capture_cfg_t
- overflows : input_capture_callback_args_t , input_capture_capture_t
- overflows_current : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- overflows_last : gpt_input_capture_instance_ctrl_t
- oversampling : sdadc_channel_cfg_t
- ovf_interrupt_enabled : cac_cfg_t
- own_addr_type : sf_ble_adv_info_t , sf_ble_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- p -

- p_address : adc_info_t
- p_aglorithm_specific_params : sf_crypto_signature_context_t
- p_api : adc_instance_t , aes_instance_t , analog_connect_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , cgc_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t ,

dsa_instance_t , ecc_instance_t , elc_instance_t , external_irq_instance_t , flash_instance_t , fmi_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , ioport_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lpmv2_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t , sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t , trng_instance_t , uart_instance_t , wdt_instance_t

- p_app_ptp_rx_desc : ptpedmac_instance_ctrl_t
- p_attr_value : sf_ble_char_attribute_t , sf_ble_svc_attribute_t
- p_auth_tag : sf_crypto_cipher_aes_init_params_t
- p_base : display_clut_cfg_t , display_input_cfg_t
- p_bit_timing : can_cfg_t
- p_ble_callback : sf_ble_provisioning_t
- p_block_array : flash_fmi_regions_t
- p_block_pool_name : sf_message_cfg_t
- p_buff : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , sf_cellular_cmd_resp_t , sf_wifi_qca4010_cmd_resp_t
- p_buffer : pdc_callback_args_t , pdc_cfg_t
- p_buffer_pool_rx : sf_wifi_cfg_t
- p_bus : sf_i2c_cfg_t , sf_i2c_instance_ctrl_t , sf_spi_cfg_t , sf_spi_instance_ctrl_t
- p_bus_name : sf_i2c_bus_t , sf_spi_bus_t
- p_buttons : sf_touch_ctsu_cfg_t
- p_callback : adc_cfg_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_cfg_t , cac_instance_ctrl_t , can_cfg_t , can_instance_ctrl_t , comparator_cfg_t , ctsu_cfg_t , ctsu_instance_ctrl_t , display_cfg_t , dmac_instance_ctrl_t , doc_cfg_t , doc_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_cfg_t , flash_cfg_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_cfg_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_cfg_t , iwdt_instance_ctrl_t , jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , keymatrix_cfg_t , lvd_cfg_t , lvd_instance_ctrl_t , pdc_cfg_t , pdc_instance_ctrl_t , ptp_cfg_t , ptp_instance_ctrl_t , ptpedmac_cfg_t , ptpedmac_instance_ctrl_t , rspi_instance_ctrl_t , rtc_cfg_t , rtc_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_cfg_t , sf_audio_playback_hw_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_el_fx_config_t , sf_el_fx_instance_ctrl_t , sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t , sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t , sf_message_buffer_ctrl_t , sf_message_post_cfg_t , sf_power_profiles_v2_low_power_cfg_t , sf_wifi_cfg_t , sf_wifi_provisioning_t , sf_wifi_wps_t , spi_cfg_t , ssi_instance_ctrl_t ,

sf_audio_playback_instance_ctrl_t , timer_cfg_t , transfer_cfg_t , uart_cfg_t , wdt_cfg_t , wdt_instance_ctrl_t

- p_callback_memory : ctsu_instance_ctrl_t
- p_callback_rec : NX_REC
- p_canvas : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_capture_data_buffer : sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t
- p_cellular_mutex : sf_cellular_instance_cfg_t
- p_cfg : adc_instance_t , aes_instance_t , analog_connect_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , cgc_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t , dsa_instance_t , ecc_instance_t , elc_instance_t , external_irq_instance_t , flash_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , ioport_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lpmv2_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_cfg_t , sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t , sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_cfg_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t , trng_instance_t , uart_instance_t , wdt_instance_t
- p_channel_cfg : adc_instance_t
- p_channel_cfgs : adc_on_sdadc_cfg_t
- p_chap_get_challenge_cb : sf_cellular_nsal_cfg_t
- p_chap_get_responder_cb : sf_cellular_nsal_cfg_t
- p_chap_get_verify_cb : sf_cellular_nsal_cfg_t
- p_char_multiple_read_rsp : sf_ble_char_read_rsp_t
- p_char_read_by_handle_rsp : sf_ble_char_read_rsp_t
- p_char_read_by_uuid_rsp : sf_ble_char_read_rsp_t
- p_chip : sf_touch_panel_v2_instance_ctrl_t
- p_chipset : sf_wifi_info_t
- p_cipher_context_buffer : sf_crypto_cipher_instance_ctrl_t
- p_circular_queue : sf_cellular_circular_queue_cfg_t
- p_circular_queue_buffer : sf_cellular_circular_queue_cfg_t
- p_circular_queue_cfg : sf_cellular_extended_cfg_t
- p_clock_cfg : sf_power_profiles_v2_run_cfg_t
- p_clut : display_clut_t
- p_cmd : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- p_cmd_param_callback : sf_cellular_cfg_t
- p_cmd_queue_ptr : sf_wifi_qca4010_queue_cfg_t
- p_cmd_set : sf_cellular_cfg_t , sf_wifi_qca4010_cfg_t
- p_common_cfg : sf_audio_playback_cfg_t
- p_common_ctrl : sf_audio_playback_cfg_t , sf_audio_playback_instance_ctrl_t
- p_comms : sf_console_cfg_t , sf_console_instance_ctrl_t

- p_config : sf_el_fx_t
- p_connection_table : analog_connect_table_t
- p_context : adc_callback_args_t , adc_cfg_t , adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_callback_args_t , cac_cfg_t , cac_instance_ctrl_t , can_callback_args_t , can_cfg_t , can_instance_ctrl_t , cgc_callback_args_t , comparator_callback_args_t , comparator_cfg_t , ctsu_callback_args_t , ctsu_cfg_t , ctsu_instance_ctrl_t , display_callback_args_t , display_cfg_t , dmac_instance_ctrl_t , doc_callback_args_t , doc_cfg_t , doc_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_callback_args_t , external_irq_cfg_t , flash_callback_args_t , flash_cfg_t , glcd_ctrl_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_callback_args_t , i2c_cfg_t , i2s_callback_args_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_callback_args_t , input_capture_cfg_t , iwdt_instance_ctrl_t , jpeg_decode_callback_args_t , jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_callback_args_t , jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , keymatrix_callback_args_t , keymatrix_cfg_t , lvd_callback_args_t , lvd_cfg_t , pdc_callback_args_t , pdc_cfg_t , pdc_instance_ctrl_t , ptp_callback_args_t , ptp_cfg_t , ptp_instance_ctrl_t , ptpedmac_callback_args_t , ptpedmac_cfg_t , ptpedmac_instance_ctrl_t , rspi_instance_ctrl_t , rtc_callback_args_t , rtc_cfg_t , rtc_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_callback_args_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t , sf_adc_periodic_callback_args_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_callback_args_t , sf_audio_playback_hw_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t , sf_cellular_callback_args_t , sf_cellular_cfg_t , sf_crypto_cfg_t , sf_crypto_instance_ctrl_t , sf_el_fx_callback_args_t , sf_el_fx_config_t , sf_el_fx_instance_ctrl_t , sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t , sf_el_lx_nor_callback_args_t , sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t , sf_message_buffer_ctrl_t , sf_message_callback_args_t , sf_message_post_cfg_t , sf_power_profiles_v2_callback_args_t , sf_power_profiles_v2_low_power_cfg_t , sf_touch_ctsu_cfg_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t , sf_touchpanel_v2_callback_args_t , sf_wifi_callback_args_t , sf_wifi_cfg_t , sf_wifi_qca4010_cfg_t , slcdc_instance_ctrl_t , spi_callback_args_t , spi_cfg_t , ssi_instance_ctrl_t , timer_callback_args_t , timer_cfg_t , transfer_callback_args_t , transfer_cfg_t , uart_callback_args_t , uart_cfg_t , wdt_callback_args_t , wdt_cfg_t , wdt_instance_ctrl_t
- p_correction_info : ctsu_instance_ctrl_t
- p_crypto_api : aes_cfg_t , aes_ctrl_t , arc4_cfg_t , arc4_ctrl_t , dsa_cfg_t , dsa_ctrl_t , ecc_cfg_t , ecc_ctrl_t , hash_cfg_t , hash_ctrl_t , key_installation_instance_ctrl_t , rsa_cfg_t , rsa_ctrl_t , tdes_cfg_t , tdes_ctrl_t , trng_cfg_t , trng_ctrl_t
- p_crypto_ctrl : aes_ctrl_t , arc4_ctrl_t , dsa_ctrl_t , ecc_ctrl_t , hash_cfg_t , key_installation_instance_ctrl_t , rsa_ctrl_t , trng_ctrl_t
- p_ctrl : adc_instance_t , aes_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t , dsa_instance_t , ecc_instance_t , external_irq_instance_t , flash_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_callback_args_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t ,

sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_fx_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t , trng_instance_t , uart_instance_t , wdt_instance_t

- p_ctsu_cfg : ctsu_instance_ctrl_t
- p_ctsu_instance : sf_touch_ctsu_cfg_t , sf_touch_ctsu_instance_ctrl_t
- p_ctsuwr : ctsu_instance_ctrl_t
- p_current_buffer : pdc_instance_ctrl_t
- p_current_menu : sf_console_instance_ctrl_t
- p_data : key_installation_key_t , r_crypto_data_handle_t , rsa_key_t , sf_audio_playback_data_t , sf_ble_char_multiple_read_rsp_t , sf_ble_char_read_by_handle_rsp_t , sf_ble_char_read_by_uuid_rsp_t , sf_ble_char_write_req_t , sf_ble_event_info_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t , sf_cellular_callback_args_t , sf_crypto_data_handle_t , sf_el_fx_media_partition_info_t , sf_wifi_callback_args_t , sf_audio_playback_instance_ctrl_t
- p_data_buffer : sf_adc_periodic_callback_args_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- p_delay_callback : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_delay_callback_context : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_dest : transfer_info_t
- p_disconnect_callback : sf_comms_telnet_cfg_t , sf_comms_telnet_instance_ctrl_t
- p_display : sf_el_gx_instance_ctrl_t
- p_display_instance : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_display_runtime_cfg : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_drift_buf : sf_touch_ctsu_button_info_t
- p_drift_count : sf_touch_ctsu_button_info_t
- p_driver_handle : sf_ble_ctrl_t , sf_cellular_ctrl_t , sf_wifi_ctrl_t , sf_wifi_qca4010_ctrl_t
- p_elem_index : sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- p_elements : ctsu_cfg_t
- p_erase_sizes_bytes : qspi_info_t
- p_eventflag : sf_wifi_qca4010_extended_cfg_t
- p_extend : adc_cfg_t , cac_cfg_t , can_cfg_t , comparator_cfg_t , crc_cfg_t , ctsu_cfg_t , display_cfg_t , external_irq_cfg_t , flash_cfg_t , i2c_cfg_t , i2s_cfg_t , input_capture_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , key_installation_cfg_t , keymatrix_cfg_t , lpmv2_cfg_t , lvd_cfg_t , opamp_cfg_t , pdc_cfg_t , ptp_cfg_t , ptp_instance_ctrl_t , qspi_cfg_t , rtc_cfg_t , sdmmc_cfg_t , sf_adc_periodic_cfg_t , sf_audio_playback_common_cfg_t , sf_audio_playback_hw_cfg_t , sf_audio_record_cfg_t , sf_ble_cfg_t , sf_ble_onboard_profile_cfg_t , sf_block_media_cfg_t , sf_cellular_cfg_t , sf_cellular_nsal_cfg_t , sf_cellular_socket_cfg_t , sf_comms_cfg_t , sf_crypto_cfg_t , sf_crypto_cipher_cfg_t , sf_crypto_hash_cfg_t , sf_crypto_key_cfg_t , sf_crypto_key_installation_cfg_t , sf_crypto_signature_cfg_t , sf_crypto_trng_cfg_t , sf_el_fx_config_t , sf_memory_cfg_t , sf_power_profiles_v2_cfg_t , sf_power_profiles_v2_low_power_cfg_t , sf_power_profiles_v2_run_cfg_t , sf_socket_cfg_t , sf_thread_monitor_instance_ctrl_t , sf_touch_ctsu_cfg_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_cfg_t , sf_wifi_cfg_t , sf_wifi_onchip_stack_cfg_t , sf_wifi_qca4010_cfg_t , sf_wifi_qca4010_socket_cfg_t , spi_cfg_t , timer_cfg_t , transfer_cfg_t , uart_cfg_t , wdt_cfg_t
- p_extpin_ctrl : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- p_framebuffer_a : sf_el_gx_cfg_t
- p_framebuffer_b : sf_el_gx_cfg_t

- p_framebuffer_read : sf_el_gx_instance_ctrl_t
- p_framebuffer_write : sf_el_gx_instance_ctrl_t
- p_fwk_common_api : sf_crypto_key_instance_ctrl_t
- p_fwk_common_ctrl : sf_crypto_key_instance_ctrl_t
- p_gamma_correction : display_output_cfg_t
- p_hal_api : sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_hal_ctrl : sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_hidden_sector : sf_el_fx_callback_args_t
- p_huffman_croma_ac_table : jpeg_encode_cfg_t
- p_huffman_croma_dc_table : jpeg_encode_cfg_t
- p_huffman_luma_ac_table : jpeg_encode_cfg_t
- p_huffman_luma_dc_table : jpeg_encode_cfg_t
- p_hysteresis : sf_touch_ctsu_button_info_t
- p_info : transfer_cfg_t
- p_initial_menu : sf_console_cfg_t
- p_interface : sf_wifi_nsal_callback_args_t
- p_ioport_pin_tbl : sf_power_profiles_v2_run_cfg_t
- p_ioport_pin_tbl_enter : sf_power_profiles_v2_low_power_cfg_t
- p_ioport_pin_tbl_exit : sf_power_profiles_v2_low_power_cfg_t
- p_ip : sf_cellular_nsal_cfg_t , sf_wifi_nsal_callback_args_t
- p_iv : sf_crypto_cipher_aes_init_params_t
- p_jpegbuffer : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_key : arc4_cfg_t
- p_key_data : sf_crypto_signature_context_t
- p_leds : bsp_leds_t
- p_link_down_cb : sf_cellular_nsal_cfg_t
- p_link_up_cb : sf_cellular_nsal_cfg_t
- p_lock_mutex : sf_i2c_bus_t , sf_spi_bus_t
- p_low_lvl_ble : sf_ble_onboard_profile_cfg_t , sf_ble_onboard_profile_ctrl_t
- p_low_lvl_sf_comms : sf_ble_on_rl78g1d_cfg_t
- p_low_lvl_timer : sf_ble_on_rl78g1d_cfg_t
- p_lower_lvl : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_lower_lvl_adc : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- p_lower_lvl_adc_periodic : sf_audio_record_adc_instance_ctrl_t
- p_lower_lvl_api : sf_i2c_bus_t , sf_spi_bus_t
- p_lower_lvl_block_media : sf_el_fx_config_t , sf_el_fx_instance_ctrl_t
- p_lower_lvl_cellular : sf_cellular_socket_cfg_t , sf_cellular_socket_ctrl_t
- p_lower_lvl_cfg : sf_i2c_cfg_t , sf_spi_cfg_t
- p_lower_lvl_common : sf_crypto_key_installation_cfg_t , sf_crypto_trng_cfg_t
- p_lower_lvl_common_api : sf_crypto_key_installation_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_common_ctrl : sf_crypto_key_installation_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_crypto : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- p_lower_lvl_crypto_api : key_installation_cfg_t
- p_lower_lvl_crypto_common : sf_crypto_cipher_cfg_t , sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_key_cfg_t , sf_crypto_signature_cfg_t
- p_lower_lvl_crypto_trng : sf_crypto_cipher_cfg_t
- p_lower_lvl_ctrl : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- p_lower_lvl_dac : sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_framewrk : sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t ,

sf_touch_panel_chip_sx8654_instance_ctrl_t
- p_lower_lvl_fwk_common_api : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_fwk_common_ctrl : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_hw : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t
- p_lower_lvl_i2c : sf_i2c_instance_ctrl_t
- p_lower_lvl_i2s : sf_audio_playback_hw_i2s_cfg_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_i2s_instance_ctrl_t
- p_lower_lvl_icu : sf_wifi_on_gt202_cfg_t
- p_lower_lvl_instance : sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_trng_cfg_t
- p_lower_lvl_irq : sf_external_irq_cfg_t , sf_external_irq_instance_ctrl_t , sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- p_lower_lvl_jpeg_decode : sf_jpeg_decode_cfg_t , sf_jpeg_decode_instance_ctrl_t
- p_lower_lvl_lpm : sf_power_profiles_v2_low_power_cfg_t
- p_lower_lvl_onchip_wifi : sf_socket_cfg_t , sf_socket_ctrl_t
- p_lower_lvl_onchip_wifi_qca4010 : sf_wifi_qca4010_socket_cfg_t , sf_wifi_qca4010_socket_ctrl_t
- p_lower_lvl_qspi : sf_block_media_qspi_instance_ctrl_t
- p_lower_lvl_sdmmc : sf_block_media_sdmmc_instance_ctrl_t
- p_lower_lvl_sf_crypto_hash : sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_sf_crypto_trng_ctrl : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_spi : sf_wifi_on_gt202_cfg_t
- p_lower_lvl_timer : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_transfer : pdc_cfg_t , pdc_instance_ctrl_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_uart : sf_uart_comms_cfg_t , sf_uart_comms_instance_ctrl_t
- p_lower_lvl_wdt : sf_thread_monitor_cfg_t , sf_thread_monitor_instance_ctrl_t
- p_lower_lvl_wifi : sf_wifi_onchip_stack_cfg_t , sf_wifi_onchip_stack_ctrl_t
- p_lower_lvl_wifi_qca4010 : sf_wifi_qca4010_onchip_stack_cfg_t , sf_wifi_qca4010_onchip_stack_ctrl_t
- p_lx_nor_flash : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_mailbox : can_cfg_t , can_instance_ctrl_t
- p_mailbox_mask : can_extended_cfg_t
- p_memory_pool : sf_crypto_cfg_t
- p_memory_settings : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_message : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t
- p_message_buffer : sf_crypto_hash_context_t
- p_message_buffer_org : sf_crypto_hash_context_t
- p_message_digest : sf_crypto_hash_context_t
- p_message_digest_org : sf_crypto_hash_context_t
- p_modifiable_cmd_set : sf_cellular_cfg_t
- p_module_extended_cfg : sf_cellular_extended_cfg_t , sf_wifi_qca4010_extended_cfg_t
- p_mul_read_req : sf_ble_char_read_req_t
- p_mutual_pri_data : ctsu_instance_ctrl_t
- p_mutual_raw : ctsu_instance_ctrl_t
- p_mutual_snd_data : ctsu_instance_ctrl_t
- p_next_buffer : sf_audio_playback_common_instance_ctrl_t
- p_nor_flash : sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t
- p_nor_flash_name : sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t
- p_off_count : sf_touch_ctsu_button_info_t

- p_on_count : sf_touch_ctsu_button_info_t
- p_owner : sf_audio_playback_instance_ctrl_t
- p_pap_generate_login : sf_cellular_nsal_cfg_t
- p_pap_verify_login : sf_cellular_nsal_cfg_t
- p_pin_cfg_data : ioport_cfg_t
- p_position : sf_touch_ctsu_slider_info_t , sf_touch_ctsu_wheel_info_t
- p_ppp : sf_cellular_nsal_cfg_t
- p_ppp_invalid_packet_cb : sf_cellular_nsal_cfg_t
- p_ppp_packet_pool : sf_cellular_nsal_cfg_t
- p_ppp_send_byte : sf_cellular_nsal_cfg_t
- p_ppp_stack : sf_cellular_nsal_cfg_t
- p_prov_callback : sf_cellular_cfg_t
- p_ptpedmac_buffer : ptpedmac_instance_ctrl_t
- p_puk_pin : sf_cellular_cfg_t
- p_qspi : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_quant_croma_table : jpeg_encode_cfg_t
- p_quant_luma_table : jpeg_encode_cfg_t
- p_queue : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t , sf_message_post_err_t , sf_message_subscriber_t
- p_queue_buffer : sf_wifi_qca4010_queue_cfg_t
- p_queue_cfg : sf_wifi_qca4010_extended_cfg_t
- p_r_uart_instance : sf_wifi_qca4010_uart_extend_cfg_t
- p_ram_buffer : sf_block_media_ram_instance_ctrl_t
- p_read_sim_pin_info_callback : sf_cellular_cfg_t
- p_recv_callback : sf_cellular_cfg_t
- p_reference : sf_touch_ctsu_button_info_t
- p_reg : adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_instance_ctrl_t , can_instance_ctrl_t , crc_instance_ctrl_t , ctsu_instance_ctrl_t , dac8_instance_ctrl_t , dac_instance_ctrl_t , dmac_instance_ctrl_t , doc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , icu_instance_ctrl_t , iwdt_instance_ctrl_t , jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , kint_instance_ctrl_t , lvd_instance_ctrl_t , opamp_instance_ctrl_t , pdc_instance_ctrl_t , ptp_instance_ctrl_t , ptpedmac_instance_ctrl_t , qspi_instance_ctrl_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , rtc_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_instance_ctrl_t , slcdc_instance_ctrl_t , ssi_instance_ctrl_t , wdt_instance_ctrl_t
- p_reg_cfg : ptp_instance_ctrl_t
- p_reg_gen : ptp_instance_ctrl_t
- p_region_info : sf_el_lx_nor_instance_ctrl_t
- p_regions_info : sf_memory_info_t
- p_remaining_string : sf_console_callback_args_t
- p_resp_buff : sf_wifi_qca4010_instance_cfg_t
- p_rx_dest : ssi_instance_ctrl_t
- p_rx_dst : sci_uart_instance_ctrl_t
- p_sce_api_interfaces : crypto_cfg_t
- p_sce_long_plg_end_callback : crypto_cfg_t
- p_self_data : ctsu_instance_ctrl_t
- p_self_raw : ctsu_instance_ctrl_t
- p_service : sf_ble_char_attribute_t
- p_sf_comms_cfg : sf_cellular_extended_cfg_t
- p_sf_comms_instance : sf_cellular_comms_extend_cfg_t
- p_sf_comms_rx_thread : sf_cellular_comms_extend_cfg_t
- p_sf_comms_rx_thread_stack : sf_cellular_comms_extend_cfg_t
- p_sf_crypto_trng_api : sf_crypto_cipher_instance_ctrl_t

- p_sf_jpeg_decode_instance : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_sim_pin : sf_cellular_cfg_t , sf_cellular_sim_pin_info_t
- p_sim_puk : sf_cellular_sim_pin_info_t
- p_sliders : sf_touch_ctsu_cfg_t
- p_socket_status_buffer : sf_cellular_instance_cfg_t
- p_src : transfer_info_t
- p_src_transfer : sf_adc_periodic_instance_ctrl_t
- p_stream : sf_audio_playback_common_instance_ctrl_t
- p_success_resp : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- p_sync_eventflag : sf_i2c_bus_t , sf_spi_bus_t
- p_thread : sf_thread_monitor_thread_counter_t
- p_threshold : sf_touch_ctsu_button_info_t , sf_touch_ctsu_slider_info_t , sf_touch_ctsu_wheel_info_t
- p_timer : i2s_cfg_t , ssi_instance_ctrl_t
- p_touch_cfg : sf_touch_ctsu_instance_ctrl_t
- p_transfer_data : sdmmc_instance_ctrl_t
- p_transfer_rx : ctsu_cfg_t , i2c_cfg_t , i2s_cfg_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , spi_cfg_t , ssi_instance_ctrl_t , uart_cfg_t
- p_transfer_tx : ctsu_cfg_t , i2c_cfg_t , i2s_cfg_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , spi_cfg_t , ssi_instance_ctrl_t , uart_cfg_t
- p_tsn_calib_regs : adc_instance_ctrl_t
- p_tsn_ctrl_regs : adc_instance_ctrl_t
- p_tuning_count : ctsu_instance_ctrl_t
- p_tuning_diff : ctsu_instance_ctrl_t
- p_tx_packet_buffer : sf_wifi_nsal_cfg_t
- p_tx_src : sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- p_uart_instance : sf_touch_ctsu_cfg_t
- p_uart_instance_objects : sf_wifi_qca4010_instance_cfg_t
- p_uart_instances : sf_wifi_qca4010_cfg_t
- p_value : sf_ble_gatt_attr_event_t
- p_wheels : sf_touch_ctsu_cfg_t
- p_wifi_mutex : sf_wifi_qca4010_instance_cfg_t
- p_wifi_nsal_cfg : sf_wifi_nsal_callback_args_t
- p_work_memory_start : sf_message_cfg_t
- page_size : qspi_instance_ctrl_t
- pageProgram : qspi_api_t
- parent_svc_handle : sf_ble_svc_attribute_t
- parity : spi_on_rspi_cfg_t , uart_cfg_t
- parse : sf_console_api_t
- partition : sf_el_fx_media_info_t
- password : sf_cellular_provisioning_t
- patch : ssp_pack_version_t
- pause : sf_audio_playback_api_t
- payload : sf_touch_panel_v2_instance_ctrl_t , sf_touchpanel_v2_callback_args_t
- payload_buffer : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- payload_length : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- payloadGet : sf_touch_panel_chip_api_t
- pckt_size : sf_cellular_socket_info_t
- pclk_div : external_irq_cfg_t
- pclka_div : cgc_system_clock_cfg_t
- pclkb_div : cgc_system_clock_cfg_t
- pclkc_div : cgc_system_clock_cfg_t
- pclkd_div : cgc_system_clock_cfg_t

- pcm_width : i2s_cfg_t
- pdp_type : sf_cellular_provisioning_t
- peer_addr : sf_ble_connect_info_t
- peer_addr_type : sf_ble_connect_info_t
- peer_ip : sf_cellular_nsal_cfg_t
- pend : sf_message_api_t
- pending_operation : sf_memory_qspi_nor_instance_ctrl_t
- pending_operation_size : sf_memory_qspi_nor_instance_ctrl_t
- period : agt_instance_ctrl_t , timer_cfg_t
- period_counts : timer_info_t
- periodic_ipl : rtc_cfg_t
- periodic_irq : rtc_instance_ctrl_t
- periodicIrqRateSet : rtc_api_t
- periodSet : timer_api_t
- peripheral : elc_link_t
- perm : RBLE_GATT_SET_PERM
- pga0 : adc_cfg_t , adc_instance_ctrl_t
- pga1 : adc_cfg_t , adc_instance_ctrl_t
- pga2 : adc_cfg_t , adc_instance_ctrl_t
- pga_available : adc_instance_ctrl_t
- phy_mode : sf_wifi_qca4010_status_t
- pin : ioport_pin_cfg_t , sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- pin_cfg : ioport_pin_cfg_t
- pin_output : comparator_cfg_t
- pin_reset : sf_cellular_extended_cfg_t , sf_wifi_on_gt202_cfg_t , sf_wifi_qca4010_extended_cfg_t
- pin_select : agt_input_capture_extend_t
- pin_slave_select : sf_wifi_on_gt202_cfg_t
- pinCfg : ioport_api_t
- pinDirectionSet : ioport_api_t
- pinEthernetModeCfg : ioport_api_t
- pinEventInputRead : ioport_api_t
- pinEventOutputWrite : ioport_api_t
- ping : sf_cellular_socket_api_t , sf_wifi_qca4010_onchip_stack_api_t
- pinRead : ioport_api_t
- pinsCfg : ioport_api_t
- pint_irq : ptpedmac_instance_ctrl_t
- pinWrite : ioport_api_t
- pixel_format : jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t
- pixelFormatGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- play : sf_audio_playback_hw_api_t
- playing : sf_audio_playback_common_instance_ctrl_t
- pll_cfg : cgc_clocks_cfg_t
- pll_div_max : bsp_feature_cgc_t
- pll_mul_max : bsp_feature_cgc_t
- pll_mul_min : bsp_feature_cgc_t
- pll_src_configurable : bsp_feature_cgc_t
- pll_state : cgc_clocks_cfg_t
- pllccr_type : bsp_feature_cgc_t
- pointer_hdl : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_LIST
- polarity : sdadc_channel_cfg_t
- polynomial : crc_cfg_t , crc_instance_ctrl_t
- portDirectionSet : ioport_api_t

- portEventInputRead : ioport_api_t
- portEventOutputWrite : ioport_api_t
- portRead : ioport_api_t
- portWrite : ioport_api_t
- post : sf_message_api_t
- power_lvl : sf_ble_set_tx_pwr_info_t
- power_supply_state : lpmv2_mcu_cfg_t
- pp_curr_bus_ctrl : sf_i2c_bus_t
- pp_curr_ctrl : sf_i2c_bus_t , sf_spi_bus_t
- pp_subscriber_group : sf_message_subscriber_list_t
- pp_subscriber_lists : sf_message_cfg_t , sf_message_instance_ctrl_t
- ppp_stack_size : sf_cellular_nsal_cfg_t
- preamble : sf_wifi_cfg_t
- preamble_length : phy_record_t
- pref_ops : sf_cellular_cfg_t
- press_val1 : sf_ble_blp_meas_info_t
- press_val2 : sf_ble_blp_meas_info_t
- press_val3 : sf_ble_blp_meas_info_t
- prevbuf : trng_ctrl_t
- pri_ref : ctsu_mutual_buf_t
- pri_sen : ctsu_mutual_buf_t
- priority : sf_audio_playback_common_cfg_t , sf_cellular_nsal_cfg_t , sf_message_post_cfg_t , sf_thread_monitor_cfg_t , sf_touch_panel_v2_cfg_t
- priority_group_a : adc_channel_cfg_t
- productFeatureGet : fmi_api_t
- productId : sf_ble_prf_dis_pnpid_t
- productInfoGet : fmi_api_t
- productVersion : sf_ble_prf_dis_pnpid_t
- profiling_mode_check : sf_thread_monitor_instance_ctrl_t
- profiling_mode_enabled : sf_thread_monitor_cfg_t , sf_thread_monitor_instance_ctrl_t
- prompt : sf_console_api_t
- prop : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_LIST
- property : sf_ble_char_discovery_rsp_t
- protocol_mode_val : sf_ble_prf_value_t
- prov_info : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- provisionGet : sf_ble_api_t
- provisioningGet : sf_cellular_api_t , sf_wifi_api_t
- provisioningSet : sf_cellular_api_t , sf_wifi_api_t , sf_wifi_qca4010_api_t
- provisionSet : sf_ble_api_t
- pulse_count_value : agt_input_capture_extend_t
- pulse_period_first_edge : agt_input_capture_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- q -

- qspi_info : sf_memory_qspi_nor_instance_ctrl_t
- quality_factor : jpeg_encode_cfg_t
- queue : sf_comms_telnet_instance_ctrl_t , sf_uart_comms_instance_ctrl_t
- queue_mem : sf_uart_comms_instance_ctrl_t

- queue_size : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- r -

- r : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- ram_buffer_size : sf_block_media_ram_instance_ctrl_t
- rand_num : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- randomNumberGenerate : sf_crypto_trng_api_t
- rate : i2c_cfg_t , sf_ble_blp_meas_info_t
- rd_index : ctsu_instance_ctrl_t
- read : adc_api_t , cac_api_t , can_api_t , flash_api_t , i2c_api_master_t , i2s_api_t , ptpedmac_api_t , qspi_api_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , sdmmc_api_t , sf_block_media_api_t , sf_comms_api_t , sf_console_api_t , sf_i2c_api_t , sf_memory_api_t , sf_spi_api_t , spi_api_t , trng_api_t , uart_api_t
- read32 : adc_api_t
- read_bytes_max : uart_info_t
- read_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- readIo : sdmmc_api_t
- readIoExt : sdmmc_api_t
- ready : sdmmc_info_t
- reason : sf_ble_disconnect_t
- ref : ctsu_self_buf_t
- ref_time : sf_ble_prf_tip_write_data_t
- reference_clock : cac_instance_ctrl_t
- reference_voltage : bsp_feature_adc_t
- refresh : wdt_api_t
- reg_id : adc_sample_state_t
- reg_status : sf_cellular_network_status_t
- region_info : sf_memory_qspi_nor_instance_ctrl_t
- remain : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- remote_ip : sf_cellular_socket_info_t
- remote_port : sf_cellular_socket_info_t
- rendering_enable : sf_el_gx_instance_ctrl_t
- repeat_area : transfer_info_t
- repetition : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , input_capture_cfg_t
- report : sf_ble_prf_hid_report_ind_t
- report_type : sf_ble_prf_hid_report_desc_t
- req_high_throughput : sf_wifi_cfg_t
- req_type : RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_WRITE_CHAR_REQ
- reserved : sf_ble_blp_meas_info_t , sf_ble_connect_info_t , sf_ble_long_attr_info_t , sf_ble_prf_cts_curr_time_t , sf_ble_prf_cts_date_time_t , sf_ble_prf_hid_report_ind_t , sf_ble_prf_htp_temp_info_t , sf_ble_prf_ndcs_time_dst_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- reserved2 : sf_ble_connect_info_t
- reserved3 : sf_ble_connect_info_t

- reset : cac_api_t , flash_api_t , i2c_api_master_t , sf_cellular_api_t , sf_i2c_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , timer_api_t , transfer_api_t
- reset_control : wdt_cfg_t
- reset_level : sf_cellular_extended_cfg_t , sf_wifi_qca4010_extended_cfg_t
- resolution : adc_cfg_t , sdadc_instance_ctrl_t
- resource_lock : glcd_ctrl_t
- resource_lock_tx_rx : riic_instance_ctrl_t , sci_spi_instance_ctrl_t
- resp : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t
- resp_buffer_length : sf_wifi_qca4010_instance_cfg_t
- resp_wait_time : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- response : sf_ble_gatt_attr_event_t
- response_required : sf_ble_write_cmd_event_data_t
- restart : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- restarted : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t , sf_i2c_instance_ctrl_t
- resume : sf_audio_playback_api_t
- retry : sf_cellular_at_cmd_set_t
- retry_count : sf_cellular_command_parameters_info_t
- retry_delay : sf_cellular_at_cmd_set_t , sf_cellular_command_parameters_info_t , sf_wifi_qca4010_at_cmd_set_t
- riic_fastplus_rise_time : bsp_feature_riic_t
- riic_std_fast_rise_time : bsp_feature_riic_t
- ringer_cp : sf_ble_prf_ringer_cp_change_t
- ringer_setting : sf_ble_prf_ringer_setting_ntf_t
- rkey_dist : sf_ble_bonding_req_ind_t , sf_ble_bonding_start_t
- rkeys : sf_ble_bonding_response_t
- rotation_angle : sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- rr_interval : sf_ble_hrp_api_hrmeas_t
- rr_interval_num : sf_ble_hrp_api_hrmeas_t
- rs485_de_pin : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- rspi_clksyn : spi_on_rspi_cfg_t
- rspi_comm : spi_on_rspi_cfg_t
- rssi : sf_ble_info_t , sf_ble_scan_t , sf_cellular_info_t , sf_cellular_network_status_t , sf_wifi_info_t , sf_wifi_scan_t
- rts : sf_wifi_cfg_t
- runApply : sf_power_profiles_v2_api_t
- rx_bd_space : NX_REC
- rx_bytes : sf_cellular_stats_t
- rx_bytes_count : sci_uart_instance_ctrl_t
- rx_dest_bytes : ssi_instance_ctrl_t
- rx_dst_bytes : sci_uart_instance_ctrl_t
- rx_edge_start : uart_on_sci_cfg_t
- rx_err : sf_cellular_stats_t
- rx_fifo_trigger : uart_on_sci_cfg_t
- rx_in_use : ssi_instance_ctrl_t
- rx_pkts : sf_wifi_stats_t
- rx_thread_priority : sf_cellular_comms_extend_cfg_t
- rx_transfer_in_progress : sci_uart_instance_ctrl_t
- rx_zero_copy : sf_wifi_nsal_cfg_t
- rxi_ipl : i2c_cfg_t , i2s_cfg_t , spi_cfg_t , uart_cfg_t
- rxi_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , ssi_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- s -

- s_addr : in_addr
- sample_clock_divisor : lvd_extend_t
- sample_count : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t
- sample_hold_mask : adc_channel_cfg_t
- sample_hold_states : adc_channel_cfg_t
- samples : sf_audio_playback_common_instance_ctrl_t
- samples_remaining : sf_audio_playback_instance_ctrl_t
- samples_total : sf_audio_playback_instance_ctrl_t
- sampleStateCountSet : adc_api_t
- sampling_freq_hz : i2s_cfg_t , i2s_info_t , ssi_instance_ctrl_t
- sampling_rate_hz : sf_audio_record_cfg_t
- SAR : dtc_reg_t
- scalarMultiplication : ecc_api_t
- scale_bits_max : sf_audio_playback_data_type_t
- scan : sf_ble_api_t , sf_wifi_api_t , sf_wifi_qca4010_api_t
- scan_cfg_mask : sdadc_instance_ctrl_t
- scan_end_b_ipl : adc_cfg_t
- scan_end_b_irq : adc_instance_ctrl_t
- scan_end_ipl : adc_cfg_t
- scan_end_irq : adc_instance_ctrl_t , sdadc_instance_ctrl_t
- scan_interval : sf_ble_cfg_t
- scan_interval_window_val : sf_ble_scps_scan_intv_change_t
- scan_mask : adc_channel_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t
- scan_mask_group_b : adc_channel_cfg_t
- scan_mode : sf_ble_scan_info_t
- scan_response_data : sf_ble_adv_info_t , sf_ble_scan_response_data_t
- scan_response_data_length : sf_ble_scan_response_data_t
- scan_trigger : sf_adc_periodic_cfg_t
- scan_window : sf_ble_cfg_t
- scanbuf : ctsu_correction_info_t
- scanCfg : adc_api_t
- scanStart : adc_api_t , ctsu_api_t , sf_touch_ctsu_api_t
- scanStatusGet : adc_api_t
- scanStop : adc_api_t
- sda_delay : i2c_cfg_t
- sdadcClockCfg : cgc_api_t
- sdadcClockDisable : cgc_api_t
- sdadcClockEnable : cgc_api_t
- sdhi_event : sdmmc_instance_ctrl_t
- sdhi_rca : sdmmc_info_t
- sdio : sdmmc_info_t
- sdio_ipl : sdmmc_cfg_t
- sdpa : ctsu_element_cfg_t
- sdramClockOutDisable : cgc_api_t
- sdramClockOutEnable : cgc_api_t
- sec : sf_ble_prf_cts_date_time_t
- sec_info : sf_ble_provisioning_t
- sec_match : rtc_alarm_time_t

- sec_mode : sf_ble_sec_info_t
- second_coefficient : ctsu_correction_info_t
- second_val : ctsu_correction_info_t
- sector_count : sdmmc_info_t
- sector_size : sdmmc_info_t
- sectorErase : qspi_api_t
- security : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_scan_t
- semaphore : sf_crypto_instance_ctrl_t , sf_el_gx_instance_ctrl_t , sf_external_irq_instance_ctrl_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag , sf_touch_panel_v2_instance_ctrl_t
- sen : ctsu_self_buf_t
- sensor_min_sampling_time : bsp_feature_adc_t
- sensors_exclusive : bsp_feature_adc_t
- service_domain : sf_cellular_network_status_t
- service_handle : sf_ble_service_discovery_rsp_t
- set_bck_with_pckb : bsp_feature_cgc_t
- setdisplayArea : slcdc_api_t
- setExtPromiscuous : ptp_api_t
- setGcmTag : aes_api_t
- setGradientLimit : ptp_api_t
- setLocalClock : ptp_api_t
- setMasterPortID : ptp_api_t
- setMessageReceptionConfig : ptp_api_t
- setMINTevent : ptp_api_t
- setTimer : ptp_api_t
- setTxPower : sf_ble_api_t
- setup : sf_el_gx_api_t
- setWorst10Values : ptp_api_t
- sf_comms_rx_thread_stack_size : sf_cellular_comms_extend_cfg_t
- shortest_pwm_signal : gpt_instance_ctrl_t , timer_on_gpt_cfg_t
- sign : dsa_api_t , ecc_api_t , rsa_api_t
- signal : gpt_input_capture_extend_t
- signal_filter : agt_input_capture_extend_t , gpt_input_capture_extend_t
- signCrt : rsa_api_t
- signFinal : sf_crypto_signature_api_t
- signUpdate : sf_crypto_signature_api_t
- sim_status : sf_cellular_sim_pin_info_t
- simIDGet : sf_cellular_api_t
- simLock : sf_cellular_api_t
- simPinSet : sf_cellular_api_t
- simUnlock : sf_cellular_api_t
- sin_addr : sockaddr , sockaddr_in
- sin_family : sockaddr , sockaddr_in
- sin_port : sf_cellular_socket_info_t , sockaddr , sockaddr_in
- sin_zero : sockaddr , sockaddr_in
- sinfo : sf_touch_ctsu_instance_ctrl_t
- size : display_clut_cfg_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_RELIABLE_WRITE , sf_el_fx_media_partition_data_info_t , sf_el_lx_nor_memory_settings_t , transfer_info_t
- size_bytes : sf_audio_playback_data_t
- size_multiplier : sf_adc_periodic_instance_ctrl_t

- skip_internal_calibration : adc_on_sdadc_cfg_t
- slave : i2c_cfg_t
- slave_busy : riic_slave_instance_ctrl_t
- slave_latency : sf_ble_cfg_t
- slaveAddressSet : i2c_api_master_t
- slcdc_clock : slcdc_cfg_t
- slcdc_clock_setting : slcdc_cfg_t
- slope_microvolts : adc_info_t , adc_instance_ctrl_t
- SM : dtc_reg_t
- snd_ref : ctsu_mutual_buf_t
- snd_sen : ctsu_mutual_buf_t
- snoop_channel : crc_snoop_cfg_t
- snoop_direction : crc_snoop_cfg_t
- snoopCfg : crc_api_t
- snoopDisable : crc_api_t
- snoopEnable : crc_api_t
- snooze_cancel_sources : lpmv2_mcu_cfg_t
- snooze_end_sources : lpmv2_mcu_cfg_t
- snooze_request_source : lpmv2_mcu_cfg_t
- snum : ctsu_element_cfg_t
- so : ctsu_element_cfg_t
- sock_type : sf_cellular_socket_info_t
- socket_create_flag : ulpgn_socket_t
- socket_queue : sf_wifi_qca4010_instance_cfg_t
- socket_recv_buff : ulpgn_socket_t
- socket_status_buffer_length : sf_cellular_instance_cfg_t
- socketConnect : sf_wifi_qca4010_socket_api_t
- socketCreate : sf_wifi_qca4010_socket_api_t
- socketDisconnect : sf_wifi_qca4010_socket_api_t
- socketRecv : sf_wifi_qca4010_socket_api_t
- sockets : sf_wifi_qca4010_socket_ctrl_t
- socketSend : sf_wifi_qca4010_socket_api_t
- socketStatusGet : sf_wifi_qca4010_socket_api_t
- sodrv_mask : bsp_feature_cgc_t
- sodrv_shift : bsp_feature_cgc_t
- softwareEventGenerate : elc_api_t
- source_clock : cgc_clock_cfg_t
- spi_mode : qspi_instance_ctrl_t
- src_addr_mode : transfer_info_t
- ssdiv : ctsu_element_cfg_t
- ssid : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_qca4010_status_t , sf_wifi_scan_t
- ssid_broadcast : sf_wifi_cfg_t
- ssl_level_keep : spi_on_rspi_cfg_t
- ssl_neg_delay : spi_on_rspi_cfg_t
- ssl_polarity : spi_on_rspi_cfg_t
- ssl_select : spi_on_rspi_cfg_t
- stage_1_gain : sdadc_channel_cfg_t
- stage_2_gain : sdadc_channel_cfg_t
- stage_num : rsa_ctrl_t
- stamp : sf_ble_blp_meas_info_t , sf_ble_prf_cts_curr_time_t , sf_ble_prf_htp_temp_info_t , sf_ble_prf_ndcs_time_dst_t
- standby_wake_sources : lpmv2_mcu_cfg_t
- start : dac_api_t , display_api_t , display_clut_cfg_t , opamp_api_t , ptp_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t ,

sf_audio_record_api_t , sf_message_instance_range_t , sf_touch_panel_v2_api_t , slcdc_api_t , timer_api_t , transfer_api_t

- start_bitmask : gpt_input_capture_instance_ctrl_t
- start_handle : sf_ble_char_discovery_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- start_hdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_SET_PERM , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , RBLE_GATT_SVC_RANGE_LIST
- start_interrupt_enabled : riic_slave_instance_ctrl_t
- start_mode : wdt_cfg_t
- startEncryption : sf_ble_api_t
- startMeasurement : cac_api_t
- startupAreaSelect : flash_api_t
- state : arc4_ctrl_t , comparator_status_t , crypto_ctrl_t , ctsu_instance_ctrl_t , display_status_t , glcd_instance_ctrl_t , ptp_cfg_t , ptp_instance_ctrl_t , sf_ble_set_tx_pwr_info_t , sf_cellular_socket_info_t , sf_el_gx_instance_ctrl_t , sf_jpeg_decode_instance_ctrl_t , sf_message_instance_ctrl_t , sf_uart_comms_instance_ctrl_t , slcdc_instance_ctrl_t
- stateGet : pdc_api_t
- station_inactivity_timeout : sf_wifi_cfg_t
- statisticsGet : sf_cellular_api_t , sf_wifi_api_t
- status : ctsu_correction_info_t , input_capture_info_t , jpeg_decode_callback_args_t , jpeg_decode_instance_ctrl_t , jpeg_encode_callback_args_t , jpeg_encode_instance_ctrl_t , lvd_callback_args_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Cfm_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Set_Data_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Set_Perm_Complete_t , rtc_info_t , sdmmc_instance_ctrl_t , sf_ble_connect_info_t , sf_ble_disconnect_t , sf_ble_sec_enc_start_ind_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t , sf_el_fx_media_ebr_info_t , sf_el_fx_media_global_open_info_t , sf_el_fx_media_mbr_info_t , sf_touch_ctsu_button_info_t , sf_audio_playback_instance_ctrl_t
- statusClear : doc_api_t , lvd_api_t , wdt_api_t
- statusGet : comparator_api_t , crypto_api_t , display_api_t , doc_api_t , flash_api_t , jpeg_decode_api_t , jpeg_encode_api_t , lvd_api_t , opamp_api_t , qspi_api_t , sf_crypto_api_t , sf_jpeg_decode_api_t , wdt_api_t
- stca_mode : ptp_instance_ctrl_t
- stca_sync_mode : ptp_cfg_t
- stop : dac_api_t , display_api_t , i2s_api_t , opamp_api_t , ptp_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_touch_panel_v2_api_t , slcdc_api_t , timer_api_t , transfer_api_t
- Stop_ActivationRequest : transfer_api_t
- stop_bitmask : gpt_input_capture_instance_ctrl_t
- stop_bits : uart_cfg_t
- stop_control : wdt_cfg_t
- stop_level : gpt_output_pin_t
- stopMeasurement : cac_api_t
- stream_end : sf_audio_playback_data_t
- subosc_state : cgc_clocks_cfg_t
- sup_timeout : sf_ble_cfg_t
- sup_to : sf_ble_connect_info_t
- supports_8_bit_mmc : bsp_feature_sdhi_t

- suppress_carry_event_callback : rtc_instance_ctrl_t
- svc_code : sb_ble_prf_ias_set_alert_t
- switches : opamp_instance_ctrl_t
- sync_edge : display_output_cfg_t
- sync_polarity : display_timing_t
- sync_width : display_timing_t
- synchronization_jump_width : can_bit_timing_cfg_t
- sys_cfg : cgc_clocks_cfg_t
- system_clock : cgc_clocks_cfg_t
- systemClockFreqGet : cgc_api_t
- systemClockGet : cgc_api_t
- systemClockSet : cgc_api_t
- systickUpdate : cgc_api_t
- SZ : dtc_reg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- t -

- tcon_de : glcd_cfg_t
- tcon_hsync : glcd_cfg_t
- tcon_vsync : glcd_cfg_t
- tei_ipl : i2c_cfg_t , spi_cfg_t , uart_cfg_t
- tei_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t
- temp_info : sf_ble_prf_htp_temp_info_ind_t
- temp_val : sf_ble_prf_htp_temp_info_t
- text : sf_ble_anp_api_new_alert_t
- text_size : sf_ble_anp_api_new_alert_t
- thread : sf_audio_playback_common_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t
- thread_counters : sf_thread_monitor_instance_ctrl_t
- threadRegister : sf_thread_monitor_api_t
- threadUnregister : sf_thread_monitor_api_t
- threshold : gamma_correction_t , sf_touch_ctsu_button_cfg_t , sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- time : rtc_alarm_time_t
- time_segment_1 : can_bit_timing_cfg_t
- time_segment_2 : can_bit_timing_cfg_t
- time_slice : slcdc_cfg_t
- time_source : sf_ble_cts_ref_time_t
- time_zone : sf_ble_cts_local_time_t
- timeout : sf_cellular_socket_info_t , wdt_cfg_t
- timeout_clocks : wdt_timeout_values_t
- timeout_mode : riic_extended_cfg , riic_instance_ctrl_t
- timeout_period_msec : sf_thread_monitor_instance_ctrl_t
- timeout_period_watchdog_clocks : sf_thread_monitor_instance_ctrl_t
- timeout_seconds : sf_wifi_wps_t
- timeout_ticks : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- timeoutGet : wdt_api_t

- timer_channel : ptp_callback_args_t
- tk_info : sf_ble_sm_tk_ind_t
- tk_key : sf_ble_sm_tk_ind_t
- tk_req_status : sf_ble_sm_tk_ind_t
- total : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- total_count : sf_el_fx_media_partition_info_t
- total_cyc : display_timing_t
- total_lines_decoded : jpeg_decode_instance_ctrl_t
- total_partitions : sf_el_fx_config_t
- total_scan_duration : sf_ble_scan_info_t
- total_size_bytes : qspi_info_t , qspi_instance_ctrl_t
- touchDataGet : sf_touch_panel_v2_api_t
- transaction_completed : sci_i2c_instance_ctrl_t
- transaction_count : riic_slave_instance_ctrl_t
- transfer_block_current : sdmmc_instance_ctrl_t
- transfer_block_size : sdmmc_instance_ctrl_t
- transfer_blocks : UX_DCD_SYNERGY_PAYLOAD_TRANSFER
- transfer_blocks_total : sdmmc_instance_ctrl_t
- transfer_dir : sdmmc_instance_ctrl_t
- transfer_flag : ptpedmac_instance_ctrl_t
- transfer_in_progress : pdc_instance_ctrl_t , sdmmc_info_t , sdmmc_instance_ctrl_t
- transfer_irq : sdmmc_instance_ctrl_t
- transfer_length_max : transfer_properties_t
- transfer_length_remaining : transfer_properties_t
- transfer_size : adc_info_t
- transfer_times : UX_DCD_SYNERGY_PAYLOAD_TRANSFER
- transfer_width : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- transmit : sf_cellular_api_t , sf_wifi_api_t
- transpktsize : sf_cellular_qctlcatm1_socket_cfg_t
- trigger : adc_cfg_t , adc_instance_ctrl_t , comparator_cfg_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_cfg_t , keymatrix_cfg_t , sdadc_instance_ctrl_t
- trigger_channel_0 : opamp_on_opamp_cfg_t
- trigger_channel_1 : opamp_on_opamp_cfg_t
- trigger_channel_2 : opamp_on_opamp_cfg_t
- trigger_channel_3 : opamp_on_opamp_cfg_t
- trigger_enabled : sdadc_instance_ctrl_t
- trigger_group_b : adc_cfg_t
- triggerSet : external_irq_api_t , keymatrix_api_t
- trim : opamp_api_t
- trim_capable : opamp_instance_ctrl_t
- trim_channel : opamp_instance_ctrl_t
- trim_input : opamp_instance_ctrl_t
- trim_state : opamp_instance_ctrl_t
- tsn_calib_available : adc_instance_ctrl_t
- tsn_calibration_available : bsp_feature_adc_t
- tsn_control_available : bsp_feature_adc_t
- tsn_ctrl_available : adc_instance_ctrl_t
- tsn_slope : bsp_feature_adc_t
- tuning : ctsu_instance_ctrl_t
- tuning_mutual_target_value : ctsu_cfg_t , ctsu_instance_ctrl_t
- tuning_self_target_value : ctsu_cfg_t , ctsu_instance_ctrl_t
- tunning_enable : ctsu_cfg_t
- tx_bytes : sf_cellular_stats_t
- tx_err : sf_cellular_stats_t , sf_wifi_stats_t

- tx_in_use : ssi_instance_ctrl_t
- tx_pkts : sf_wifi_stats_t
- tx_power : sf_wifi_cfg_t
- tx_src_bytes : sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- tx_timeout : sf_cellular_socket_info_t
- tx_zero_copy : sf_wifi_nsal_cfg_t
- txi_ipl : i2c_cfg_t , i2s_cfg_t , spi_cfg_t , uart_cfg_t
- txi_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- type : can_frame_t , RBLE_GATT_EVENT , sf_audio_playback_data_t , sf_ble_prf_htp_temp_info_t
- tz_upd_mode : sf_cellular_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- u -

- uart_comm_mode : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- uart_rs485_mode : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- underflow_1_ipl : display_cfg_t
- underflow_2_ipl : display_cfg_t
- uniqueIdGet : fmi_api_t
- unit : adc_callback_args_t , adc_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t , timer_cfg_t
- unlock : sf_comms_api_t , sf_crypto_api_t , sf_i2c_api_t , sf_spi_api_t
- unread_count : sf_ble_anp_api_unread_alert_t
- unused : analog_connect_cfg_t
- update_bd_addr : sf_ble_cfg_t
- update_hz : sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- update_result : sf_ble_prf_rtus_time_updt_state_t
- update_state : sf_ble_prf_tip_write_data_t
- updateAnnounceFlags : ptp_api_t
- updateAnnounceMsgs : ptp_api_t
- updateClockID : ptp_api_t
- updateDelayMsgInterval : ptp_api_t
- updateDomainNumber : ptp_api_t
- updateFlashClockFreq : flash_api_t
- updateHash : hash_api_t
- updateSyncAnnounceMsgInterval : ptp_api_t
- usbClockCfg : cgc_api_t
- username : sf_cellular_provisioning_t
- uuid : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_DESC_128_LIST , RBLE_GATT_CHAR_LIST , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , sf_ble_char_desc_discovery_rsp_t , sf_ble_char_discovery_req_t , sf_ble_char_discovery_rsp_t , sf_ble_char_read_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- uuid128 : sf_ble_uuid_t
- uuid16 : sf_ble_uuid_t
- uuid32 : sf_ble_uuid_t
- uuid_length : sf_ble_uuid_t

- ux_dcd_synergy_D0_fifo_state : UX_DCD_SYNERGY
- ux_dcd_synergy_D1_fifo_state : UX_DCD_SYNERGY
- ux_dcd_synergy_ep_slave_transfer_request_semaphore : UX_DCD_SYNERGY_ED
- ux_dcd_synergy_pipe : UX_DCD_SYNERGY
- ux_synergy_next_available_bufnum : UX_HCD_SYNERGY

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- v -

- val_hdl : RBLE_GATT_SET_DATA
- val_len : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_SET_DATA , RBLE_GATT_WRITE_CHAR_REQ , sf_ble_long_attr_info_t
- valid_opamps : opamp_instance_ctrl_t
- value : RBLE_GATT_DESIRED_TYPE , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_QUERY_RESULT , RBLE_GATT_RELIABLE_WRITE , RBLE_GATT_SET_DATA , RBLE_GATT_UUID_TYPE , RBLE_GATT_WRITE_CHAR_REQ , RBLE_GATT_WRITE_RELIABLE_REQ , sf_ble_long_attr_info_t , sf_ble_prf_hid_report_desc_t
- value_handle : sf_ble_char_discovery_rsp_t
- value_size : RBLE_GATT_DESIRED_TYPE , RBLE_GATT_UUID_TYPE , sf_ble_prf_hid_report_desc_t
- variant : gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , input_capture_info_t
- vendorId : sf_ble_prf_dis_pnpid_t
- vendorIdSource : sf_ble_prf_dis_pnpid_t
- verify : dsa_api_t , ecc_api_t , rsa_api_t
- verifyFinal : sf_crypto_signature_api_t
- verifyUpdate : sf_crypto_signature_api_t
- version : sf_ble_chipset_info_t , sf_i2c_api_t , sf_spi_api_t , sf_wifi_ip_addr_t , sf_wifi_qca4010_ip_addr_t
- version_id : ssp_pack_version_t , ssp_version_t
- versionGet : adc_api_t , aes_api_t , analog_connect_api_t , arc4_api_t , cac_api_t , can_api_t , cgc_api_t , comparator_api_t , crc_api_t , crypto_api_t , ctsu_api_t , dac_api_t , display_api_t , doc_api_t , dsa_api_t , ecc_api_t , elc_api_t , external_irq_api_t , flash_api_t , fmi_api_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , input_capture_api_t , ioport_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , lpmv2_api_t , lvd_api_t , opamp_api_t , pdc_api_t , ptp_api_t , ptpedmac_api_t , qspi_api_t , rsa_api_t , rtc_api_t , sdmmc_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t ,

sf_el_gx_api_t , sf_external_irq_api_t , sf_jpeg_decode_api_t , sf_memory_api_t , sf_message_api_t , sf_power_profiles_v2_api_t , sf_socket_api_t , sf_thread_monitor_api_t , sf_touch_ctsu_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t , wdt_api_t

- vertical_resolution : jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , jpeg_encode_raw_image_parameters
- voltage_ref : adc_cfg_t , adc_instance_ctrl_t
- voltage_slope : lvd_cfg_t
- voltage_threshold : lvd_cfg_t
- volume : sf_audio_playback_common_instance_ctrl_t
- volumeSet : sf_audio_playback_api_t
- vref_src : adc_on_sdadc_cfg_t
- vref_voltage : adc_on_sdadc_cfg_t
- vsize : display_input_cfg_t , glcd_ctrl_t
- vsize_pixels : sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- vsync : pdc_state_t
- vsync_polarity : pdc_cfg_t , pdc_instance_ctrl_t
- vtiming : display_output_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- w -

- wait : sf_external_irq_api_t , sf_jpeg_decode_api_t
- wait_option : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- wave_form : slcdc_cfg_t
- wds : sf_wifi_cfg_t
- wdt_open : wdt_instance_ctrl_t
- whitelistAdd : sf_ble_api_t
- whitelistDel : sf_ble_api_t
- wifiStatusGet : sf_wifi_qca4010_api_t
- window_end : wdt_cfg_t
- window_start : wdt_cfg_t
- winfo : sf_touch_ctsu_instance_ctrl_t
- wmm : sf_wifi_cfg_t
- word_length : i2s_cfg_t
- work_buffer : aes_ctrl_t
- work_memory_size_bytes : sf_message_cfg_t
- wps_key : sf_wifi_wps_t
- wps_mode : sf_wifi_wps_t
- wpsStart : sf_wifi_api_t
- wr_index : ctsu_instance_ctrl_t
- wr_offset : RBLE_GATT_WRITE_CHAR_REQ
- write : can_api_t , dac_api_t , doc_api_t , flash_api_t , i2c_api_master_t , i2s_api_t , sdmmc_api_t , sf_block_media_api_t , sf_comms_api_t , sf_console_api_t , sf_i2c_api_t , sf_memory_api_t , sf_spi_api_t , slcdc_api_t , spi_api_t , uart_api_t

- write_bytes_max : uart_info_t
- write_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- write_protect : sdmmc_extended_cfg_t , sdmmc_instance_ctrl_t
- write_protected : sdmmc_info_t
- writeIo : sdmmc_api_t
- writeIoExt : sdmmc_api_t
- writeRead : i2s_api_t , sf_spi_api_t , spi_api_t
- ws_continue : i2s_cfg_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- x -

- x : display_coordinate_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_payload_t
- x_capture_pixels : pdc_cfg_t , pdc_instance_ctrl_t
- x_capture_start_pixel : pdc_cfg_t , pdc_instance_ctrl_t
- x_resolution_pixels : pdc_instance_ctrl_t
- xip_mode : qspi_instance_ctrl_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- y -

- y : display_coordinate_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_payload_t
- y_capture_pixels : pdc_cfg_t , pdc_instance_ctrl_t
- y_capture_start_pixel : pdc_cfg_t , pdc_instance_ctrl_t
- y_resolution_pixels : pdc_instance_ctrl_t
- year : sf_ble_prf_cts_date_time_t
- year_match : rtc_alarm_time_t

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- z -

- zeroPaddingDecrypt : aes_api_t
- zeroPaddingEncrypt : aes_api_t
- zeros_written : ssi_instance_ctrl_t

6.3.2 Functions

- BSP_ALIGN_VARIABLE_V2() : ptpedmac_instance_ctrl_t , sf_audio_playback_common_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t

6.3.3 Variables

- a -

- a : display_color_t
- absolute_start_addr : sf_el_lx_nor_memory_settings_t
- accept : sf_ble_bonding_response_t
- accept_addr : sf_ble_addr_verify_ind_t
- access_control : sf_wifi_cfg_t
- access_delay : spi_on_rspi_cfg_t
- access_ipl : sdmmc_cfg_t
- access_tech_name : sf_cellular_network_status_t
- accessWindowClear : flash_api_t
- accessWindowSet : flash_api_t
- accuracy : sf_ble_cts_ref_time_t
- ACLAdd : sf_wifi_api_t
- ACLDelete : sf_wifi_api_t
- activation_on_rxi : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- activation_on_txi : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- activation_source : transfer_cfg_t
- active : sf_thread_monitor_thread_counter_t
- active_band : sf_cellular_network_status_t
- actual_count : sf_el_fx_media_partition_info_t
- actual_hwErr_event : riic_instance_ctrl_t
- ad_da_synchronized : dac_cfg_t
- adc_calib_available : adc_instance_ctrl_t
- add_average_count : adc_cfg_t
- add_mask : adc_channel_cfg_t
- addAdditionalAuthenticationData : aes_api_t
- addition_supported : bsp_feature_adc_t
- addr : sf_ble_addr_t , sf_wifi_ip_addr_t , sf_wifi_qca4010_ip_addr_t
- addr_high : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_loaded : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_low : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_mode : i2c_cfg_t
- addr_remain : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_total : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- addr_type : sf_ble_addr_verify_ind_t , sf_ble_connection_t , sf_ble_scan_t

- address : ptp_instance_ctrl_t
- address_restarted : riic_instance_ctrl_t
- address_type : sf_ble_scan_info_t
- adjust_reason : sf_ble_prf_cts_curr_time_t
- adjustment_mode : rtc_error_adjustment_mode_cfg_t
- adjustment_period : rtc_error_adjustment_mode_cfg_t
- adjustment_type : rtc_error_adjustment_cfg_t
- adjustment_value : rtc_error_adjustment_cfg_t
- adv_chnl_map : sf_ble_adv_info_t
- adv_data : sf_ble_adv_info_t
- adv_data_length : sf_ble_adv_data_t
- adv_filt_policy : sf_ble_adv_info_t
- adv_intv_max : sf_ble_adv_info_t
- adv_intv_min : sf_ble_adv_info_t
- adv_type : sf_ble_adv_info_t
- advertisementStart : sf_ble_api_t
- advertisementStop : sf_ble_api_t
- agt_link : opamp_on_opamp_cfg_t
- agtio_output_enabled : timer_on_agt_cfg_t
- agto_output_enabled : timer_on_agt_cfg_t
- agtoa_output_enable : timer_on_agt_cfg_t
- agtob_output_enable : timer_on_agt_cfg_t
- airplane_mode : sf_cellular_provisioning_t
- alarm_ipl : rtc_cfg_t
- alarm_irq : rtc_instance_ctrl_t
- alert : sf_ble_anp_api_unread_alert_ntf_t
- alert_lvl : sf_ble_prf_ias_alert_lvl_change_t
- alert_num : sf_ble_anp_api_new_alert_t
- alert_status : sf_ble_prf_alert_status_ntf_t
- aligned_buff : sdmmc_instance_ctrl_t
- alignment : adc_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t
- alpha_value : jpeg_decode_cfg_t
- api_version_major : ssp_version_t
- api_version_minor : ssp_version_t
- apn : sf_cellular_provisioning_t
- arc4Process : arc4_api_t
- argumentFind : sf_console_api_t
- att_code : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Cancel_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Discovery_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Reliable_Resp_t , RBLE_GATT_WRITE_RESP

- att_hdl : RBLE_GATT_WRITE_RESP
- attr_declare_handle : sf_ble_char_attribute_t
- attr_declare_type : sf_ble_char_attribute_t
- attr_handle : sf_ble_gatt_attr_event_t , sf_ble_svc_attribute_t
- attr_hdl : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_DESC_128_LIST , RBLE_GATT_CHAR_DESC_LIST , RBLE_GATT_CHAR_LIST , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , sf_ble_long_attr_info_t
- attr_perm : sf_ble_char_attribute_t
- attr_properties : sf_ble_char_attribute_t
- attr_type : sf_ble_svc_attribute_t
- attr_uuid : sf_ble_char_attribute_t
- attr_value_handle : sf_ble_char_attribute_t
- attr_value_len : sf_ble_char_attribute_t , sf_ble_svc_attribute_t
- atune1 : ctsu_cfg_t
- audio_clk_freq_hz : i2s_cfg_t
- audio_clock : i2s_on_ssi_cfg_t
- auth_req : sf_ble_bonding_req_ind_t
- auth_type : sf_ble_sec_enc_start_ind_t , sf_cellular_nsal_cfg_t , sf_cellular_provisioning_t
- authorization : sf_ble_api_t
- auto_enable : transfer_cfg_t
- auto_execute : RBLE_GATT_WRITE_CHAR_REQ , RBLE_GATT_WRITE_RELIABLE_REQ , sf_ble_char_write_req_t
- autoClearEvent : ptp_api_t
- autostart : elc_cfg_t , external_irq_cfg_t , input_capture_cfg_t , keymatrix_cfg_t , sf_console_cfg_t , timer_cfg_t , wdt_cfg_t
- available : sf_comms_telnet_instance_ctrl_t
- average : ctsu_instance_ctrl_t , sdadc_channel_cfg_t

- b -

- b : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- back_porch : display_timing_t , glcd_ctrl_t
- bankSelect : qspi_api_t
- base_addr : sf_el_fx_media_ebr_info_t
- batt_lvl : sf_ble_bas_battery_lvl_ntf_t
- baud_rate : uart_cfg_t
- baud_rate_error_x_1000 : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- baud_rate_prescaler : can_bit_timing_cfg_t
- baudclk_out : uart_on_sci_cfg_t
- baudSet : uart_api_t
- bcast_mode : sf_ble_provisioning_t
- bclk_div : cgc_system_clock_cfg_t
- bd_addr : sf_ble_addr_verify_ind_t , sf_ble_bonding_req_ind_t , sf_ble_cfg_t , sf_ble_chipset_info_t , sf_ble_connection_t , sf_ble_scan_t
- bd_buffer_ptr : EMAC_BD
- bd_bufsize : EMAC_BD
- bd_nx_packet : EMAC_BD
- bd_rxdatalength : EMAC_BD

- bd_status : EMAC_BD
- beacon : sf_wifi_cfg_t
- ber : sf_cellular_info_t
- bg_color : display_layer_t , display_output_cfg_t
- bias_method : slcdc_cfg_t
- binfo : sf_touch_ctsu_instance_ctrl_t
- bit_order : crc_cfg_t , crc_instance_ctrl_t , spi_cfg_t
- bit_width : dac_info_t
- bitrate : spi_cfg_t
- bitrate_modulation : sci_i2c_extended_cfg , sci_spi_extended_cfg , sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- blankCheck : flash_api_t
- ble_driver_thread_priority : sf_ble_on_rl78g1d_cfg_t
- ble_prf_value : sf_ble_prf_hid_change_event_t
- ble_serial_thread_priority : sf_ble_on_rl78g1d_cfg_t
- block_pool : sf_message_instance_ctrl_t
- block_section_end_addr : flash_fmi_block_info_t
- block_section_st_addr : flash_fmi_block_info_t
- block_size : flash_fmi_block_info_t , sdmmc_extended_cfg_t , sf_block_media_cfg_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_block_media_ram_instance_ctrl_t , sf_block_media_sdmmc_instance_ctrl_t
- block_size_write : flash_fmi_block_info_t
- blockReset : transfer_api_t
- bonding_mode : sf_ble_provisioning_t
- bonding_status : sf_ble_sec_enc_start_ind_t
- bondingResponse : sf_ble_api_t
- bondingStart : sf_ble_api_t
- boot_record_table : sf_el_fx_media_info_t
- brightness : display_correction_t , display_output_cfg_t
- bss_type : sf_wifi_scan_t
- bssid : sf_wifi_qca4010_scan_t , sf_wifi_scan_t
- buff : sf_el_fx_media_ebr_info_t , sf_el_fx_media_mbr_info_t
- buff_len : sf_cellular_cmd_resp_t , sf_wifi_qca4010_cmd_resp_t
- buffer : sf_crypto_signature_context_t
- buffer_index : sf_adc_periodic_callback_args_t , sf_audio_playback_common_instance_ctrl_t
- buffer_keep : sf_message_acquire_cfg_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- buffer_size : sf_audio_record_i2s_instance_ctrl_t , sf_message_cfg_t , sf_message_instance_ctrl_t
- bufferAcquire : sf_message_api_t
- bufferRelease : sf_message_api_t
- build : ssp_pack_version_t
- bus_width : sdmmc_hw_t , sdmmc_info_t
- busClockOutCfg : cgc_api_t
- busClockOutDisable : cgc_api_t
- busClockOutEnable : cgc_api_t
- byte_pool : sf_crypto_instance_ctrl_t
- byte_swap : spi_on_rspi_cfg_t
- bytes : i2c_callback_args_t , sf_console_callback_args_t
- bytes_per_pixel : pdc_cfg_t , pdc_instance_ctrl_t , sf_el_gx_instance_ctrl_t

- c -

- cac_api_open : cac_instance_ctrl_t
- cac_continous_mode : cac_instance_ctrl_t
- cac_lock : cac_instance_ctrl_t
- cac_lower_limit : cac_cfg_t
- cac_meas_clock : cac_cfg_t
- cac_ref_clock : cac_cfg_t
- cac_upper_limit : cac_cfg_t
- cache_state : flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t
- calculate : crc_api_t
- calendarAlarmGet : rtc_api_t
- calendarAlarmSet : rtc_api_t
- calendarCounterStart : rtc_api_t
- calendarCounterStop : rtc_api_t
- calendarTimeGet : rtc_api_t
- calendarTimeSet : rtc_api_t
- calib_adc_skip : adc_cfg_t
- calib_end_irq : sdadc_instance_ctrl_t
- calib_status : sdadc_instance_ctrl_t
- calibrate : adc_api_t , sf_touch_panel_v2_api_t , sf_touch_panel_v2_instance_ctrl_t
- calibration_data : adc_info_t
- calibration_end_ipl : adc_on_sdadc_cfg_t
- calibration_ongoing : adc_info_t
- calibration_reg_available : bsp_feature_adc_t
- callback : adc_instance_ctrl_t , sf_console_command_t
- callback_used : sf_external_irq_instance_ctrl_t
- callbackSet : ctsu_api_t , sf_touch_ctsu_api_t
- cancel_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- canvasInit : sf_el_gx_api_t
- cap : ctsu_cfg_t
- capture_count : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- capture_data_buffer_size : sf_audio_record_cfg_t
- capture_data_size : sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t
- capture_irq : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- capture_irq_ipl : input_capture_cfg_t
- captureStart : pdc_api_t
- card_detect : sdmmc_extended_cfg_t
- card_ipl : sdmmc_cfg_t
- card_type : sdmmc_info_t
- carry_ipl : rtc_cfg_t
- carry_irq : rtc_instance_ctrl_t
- carry_isr_triggered : rtc_instance_ctrl_t
- category_id : sf_ble_anp_ancp_t , sf_ble_anp_api_new_alert_t , sf_ble_anp_api_unread_alert_t
- cccd_val : sf_ble_onboard_profile_cccd_changed_t
- celr_stats : sf_cellular_instance_cfg_t
- cf_block_size_write : bsp_feature_flash_hp
- cf_macro_size : bsp_feature_flash_lp
- cfgGet : wdt_api_t
- chain_mode : transfer_info_t
- channel : adc_callback_args_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , can_callback_args_t , can_cfg_t , can_instance_ctrl_t , comparator_callback_args_t , comparator_cfg_t , dac8_instance_ctrl_t , dac_cfg_t , dac_instance_ctrl_t , dmac_instance_ctrl_t , external_irq_callback_args_t , external_irq_cfg_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_cfg_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_callback_args_t , input_capture_cfg_t , NX_REC ,

opamp_trim_args_t , ptpedmac_callback_args_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_calibrate_args_t , sdmmc_hw_t , sf_i2c_bus_t , sf_spi_bus_t , sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_qca4010_status_t , sf_wifi_scan_t , spi_callback_args_t , spi_cfg_t , ssi_instance_ctrl_t , timer_cfg_t , transfer_on_dmac_cfg_t , uart_callback_args_t , uart_cfg_t

- channel_opened : dac8_instance_ctrl_t , dac_instance_ctrl_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t
- channel_started : dac8_instance_ctrl_t , dac_instance_ctrl_t
- channels : keymatrix_callback_args_t , keymatrix_cfg_t , kint_instance_ctrl_t
- char_code : sf_ble_blp_meas_recv_data_t , sf_ble_onboard_profile_cccd_changed_t
- char_handle : sf_ble_char_discovery_rsp_t
- char_read_type : sf_ble_char_read_req_t
- char_write_type : sf_ble_char_write_req_t
- charhdl : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_INDICATE_REQ , RBLE_GATT_NOTIFY_REQ , RBLE_GATT_WRITE_CHAR_REQ
- check_pclkb_ratio : bsp_feature_can_t
- checkINFABTstatus : ptp_api_t
- checkWorst10Values : ptp_api_t
- chip_select : sf_spi_cfg_t , sf_spi_instance_ctrl_t
- chip_select_level_active : sf_spi_cfg_t , sf_spi_instance_ctrl_t
- chipset : sf_ble_info_t , sf_cellular_info_t
- CHNE : dtc_reg_t
- CHNS : dtc_reg_t
- cid : sf_cellular_network_status_t
- cipher_algorithm_type : sf_crypto_cipher_instance_ctrl_t
- cipher_chaining_mode : sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t
- cipherAadUpdate : sf_crypto_cipher_api_t
- cipherFinal : sf_crypto_cipher_api_t
- cipherInit : sf_crypto_cipher_api_t
- cipherUpdate : sf_crypto_cipher_api_t
- class_code : sf_message_header_t
- class_instance : sf_audio_playback_cfg_t , sf_message_header_t , sf_audio_playback_instance_ctrl_t
- clearINFABTstatus : ptp_api_t
- clearing : adc_cfg_t
- clearIOKeep : lpmv2_api_t
- clk_accuracy : sf_ble_connect_info_t
- clk_phase : spi_cfg_t
- clk_polarity : spi_cfg_t
- clk_src : uart_on_sci_cfg_t
- clksrc : glcd_cfg_t
- clock : bsp_feature_can_t , bsp_feature_rspi_t , bsp_feature_sci_t , cac_meas_clock_config_t , cac_ref_clock_config_t
- clock_delay : spi_on_rspi_cfg_t
- clock_div_ratio : glcd_cfg_t
- clock_divider : agt_input_capture_extend_t , gpt_input_capture_extend_t
- clock_division : pdc_cfg_t , wdt_cfg_t
- clock_frequency : timer_info_t
- clock_frequency_hz : wdt_timeout_values_t
- clock_rate : sdmmc_info_t
- clock_source : bsp_feature_adc_t , can_extended_cfg_t , can_instance_ctrl_t , rtc_cfg_t , rtc_info_t , rtc_instance_ctrl_t
- clockCheck : cgc_api_t

- clockOutCfg : cgc_api_t
- clockOutDisable : cgc_api_t
- clockOutEnable : cgc_api_t
- clocksCfg : cgc_api_t
- clockStart : cgc_api_t
- clockStop : cgc_api_t
- close : adc_api_t , aes_api_t , arc4_api_t , cac_api_t , can_api_t , comparator_api_t , crc_api_t , crypto_api_t , ctsu_api_t , dac_api_t , display_api_t , doc_api_t , dsa_api_t , ecc_api_t , external_irq_api_t , flash_api_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , input_capture_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , lvd_api_t , opamp_api_t , pdc_api_t , ptp_api_t , ptpedmac_api_t , qspi_api_t , rsa_api_t , rtc_api_t , sdmmc_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t , sf_el_gx_api_t , sf_external_irq_api_t , sf_i2c_api_t , sf_jpeg_decode_api_t , sf_memory_api_t , sf_message_api_t , sf_power_profiles_v2_api_t , sf_socket_api_t , sf_spi_api_t , sf_thread_monitor_api_t , sf_touch_ctsu_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t
- close_option : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- clut : display_api_t
- cmd_index : sf_cellular_command_parameters_info_t
- code : sf_message_header_t
- code_flash : flash_info_t
- code_version_major : ssp_version_t
- code_version_minor : ssp_version_t
- coefficient_m : sdadc_channel_cfg_t
- coefficient_n : sdadc_channel_cfg_t
- color_num : display_clut_t
- color_order : display_output_cfg_t
- color_space : jpeg_decode_cfg_t
- command : sf_console_command_t
- command_flag : sf_wifi_qca4010_instance_cfg_t
- command_id : sf_ble_anp_ancp_t
- command_list : sf_console_menu_t
- commandSend : sf_cellular_api_t
- CommandSend : sf_wifi_qca4010_api_t
- common_instance_mutex : sf_audio_playback_common_instance_ctrl_t
- communicationAbort : uart_api_t
- con_interval : sf_ble_cfg_t , sf_ble_connect_info_t
- con_latency : sf_ble_connect_info_t
- configure : ptp_api_t , rtc_api_t
- conhdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Cancel_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t ,

RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Discovery_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Char_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Reliable_Resp_t , RBLE_GATT_EXE_WR_CHAR_REQ , RBLE_GATT_INDICATE_REQ , RBLE_GATT_NOTIFY_REQ , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_WRITE_CHAR_REQ , RBLE_GATT_WRITE_RELIABLE_REQ , RBLE_GATT_WRITE_RESP , sf_ble_anp_ancp_change_t , sf_ble_anp_api_new_alert_ntf_t , sf_ble_anp_api_unread_alert_ntf_t , sf_ble_bas_battery_lvl_ntf_t , sf_ble_blp_meas_recv_data_t , sf_ble_connect_info_t , sf_ble_cts_curr_time_ntf_t , sf_ble_disconnect_t , sf_ble_hrp_api_meas_ntf_t , sf_ble_hrp_cp_change_t , sf_ble_prf_alert_status_ntf_t , sf_ble_prf_hid_change_event_t , sf_ble_prf_hid_report_ind_t , sf_ble_prf_htp_temp_info_ind_t , sf_ble_prf_ias_alert_lvl_change_t , sf_ble_prf_ringer_cp_change_t , sf_ble_prf_ringer_setting_ntf_t , sf_ble_scps_scan_intv_change_t , sf_ble_sec_enc_start_ind_t , sf_ble_sm_tk_info_t , sf_ble_tip_cp_change_t , st_sf_ble_prf_htp_meas_intv_val_t

- conn_handle : sf_ble_onboard_profile_cccd_changed_t
- conn_idx : sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- conn_mode : sf_ble_adv_info_t
- conn_timeout : sf_cellular_socket_info_t
- connect : analog_connect_api_t , sf_ble_api_t
- connectMultiple : analog_connect_api_t
- context : sf_console_callback_args_t , sf_console_command_t
- context_id : sf_cellular_provisioning_t
- contextInit : sf_crypto_signature_api_t
- continuous_mode : cac_cfg_t
- contrast : display_correction_t , display_output_cfg_t
- contrastDecrease : slcdc_api_t
- contrastIncrease : slcdc_api_t
- control : can_api_t , sdmmc_api_t
- control_point_val : sf_ble_prf_value_t
- control_point_value : sf_ble_anp_ancp_change_t , sf_ble_hrp_cp_change_t , sf_ble_tip_cp_change_t
- conv_end_irq : sdadc_instance_ctrl_t
- conversion_end_ipl : adc_on_sdadc_cfg_t
- coordinate : display_layer_t
- correction : display_api_t
- correction_proc_order : glcd_cfg_t
- count_direction : timer_info_t
- count_edge : agt_input_capture_extend_t
- count_formula : sdadc_channel_cfg_t
- count_source : agt_input_capture_extend_t , timer_on_agt_cfg_t
- counter : input_capture_callback_args_t , input_capture_capture_t ,

sf_el_fx_media_global_open_info_t
- counterGet : timer_api_t , wdt_api_t
- countIncrement : sf_thread_monitor_api_t
- country_code : sf_cellular_network_status_t
- CRA : dtc_reg_t
- CRA_b : dtc_reg_t
- CRAH : dtc_reg_t
- CRAL : dtc_reg_t
- CRB : dtc_reg_t
- crcResultGet : crc_api_t
- createKey : aes_api_t
- crossing_detected : lvd_status_t
- crypto_ctrl : tdes_ctrl_t
- csd_version : sdmmc_info_t
- csrk_key : sf_ble_sec_info_t
- ctsrts_en : uart_cfg_t
- ctsu_clock : ctsu_correction_info_t
- ctsuchac0 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac1 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac2 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac3 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac4 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchac_register_count : bsp_feature_ctsu_t
- ctsuchtrc0 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc1 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc2 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc3 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc4 : ctsu_cfg_t , ctsu_instance_ctrl_t
- ctsuchtrc_register_count : bsp_feature_ctsu_t
- ctsucr0_mask : bsp_feature_ctsu_t
- ctsucr1 : ctsu_instance_ctrl_t
- ctsucr1_mask : bsp_feature_ctsu_t
- ctsumch0_mask : bsp_feature_ctsu_t
- ctsumch1_mask : bsp_feature_ctsu_t
- ctsuso0 : ctsu_ctsuwr_t
- ctsuso1 : ctsu_ctsuwr_t
- ctsussc : ctsu_ctsuwr_t
- ctsuwr : ctsu_correction_info_t
- curr_cmd_port : sf_wifi_qca4010_instance_cfg_t
- curr_data_port : sf_wifi_qca4010_instance_cfg_t
- curr_socket_index : sf_wifi_qca4010_instance_cfg_t
- currbuf : trng_ctrl_t
- current_buffer_index : sf_audio_record_i2s_instance_ctrl_t
- current_count : sf_thread_monitor_thread_counter_t
- current_sample_count : sf_adc_periodic_instance_ctrl_t
- current_slave : rspi_instance_ctrl_t
- current_state : lvd_status_t , sf_ble_prf_rtus_time_updt_state_t
- current_time : sf_ble_cts_curr_time_ntf_t , sf_ble_prf_tip_write_data_t

- d -

- dac_mode : dac8_extended_cfg_t
- DAR : dtc_reg_t
- data : can_frame_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Resp_t , RBLE_GATT_INFO_DATA , sf_ble_adv_data_t , sf_ble_attr_info_t , sf_ble_scan_t , uart_callback_args_t
- data_bits : uart_cfg_t
- data_buffer_index : sf_adc_periodic_instance_ctrl_t
- data_buffer_length : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- data_bytes : sci_uart_instance_ctrl_t
- data_enable_polarity : display_output_cfg_t
- data_flash : flash_info_t
- data_flash_bgo : flash_cfg_t
- data_format : dac8_instance_ctrl_t , dac_cfg_t
- data_len : sf_ble_char_multiple_read_rsp_t , sf_ble_char_read_by_handle_rsp_t , sf_ble_char_read_by_uuid_rsp_t , sf_ble_scan_t
- data_length : key_installation_key_t , r_crypto_data_handle_t , sf_ble_char_write_req_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t , sf_crypto_data_handle_t
- data_length_code : can_frame_t
- data_lines : qspi_instance_ctrl_t
- data_size : sf_audio_record_i2s_instance_ctrl_t
- data_state : sf_cellular_socket_info_t
- data_type : sf_audio_playback_common_instance_ctrl_t
- dataGet : ctsu_api_t , sf_touch_ctsu_api_t
- dataTypeGet : sf_audio_playback_hw_api_t
- dave2d_buffer_cache_enabled : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- day : sf_ble_prf_cts_date_time_t
- day_of_week : sf_ble_prf_cts_curr_time_t
- dayofweek_match : rtc_alarm_time_t
- days_since_update : sf_ble_cts_ref_time_t
- decrypt : aes_api_t , rsa_api_t , tdes_api_t
- decryptCrt : rsa_api_t
- deep_standby_cancel_edge : lpmv2_mcu_cfg_t
- deep_standby_cancel_source : lpmv2_mcu_cfg_t
- delay : ptp_cfg_t , ptp_instance_ctrl_t
- desc_handle : sf_ble_char_desc_discovery_rsp_t
- desc_hdl : RBLE_GATT_CHAR_DESC_LIST
- desired_char : RBLE_GATT_DISC_CHAR_REQ
- desired_svc : RBLE_GATT_DISC_SVC_REQ
- dest_addr_mode : transfer_info_t
- detection_response : lvd_cfg_t
- dev_state : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- device : ptp_cfg_t , ptp_instance_ctrl_t , sf_el_gx_callback_args_t
- device_count : sf_i2c_bus_t , sf_spi_bus_t
- device_count_mutex : sf_i2c_bus_t , sf_spi_bus_t
- device_type : sdmmc_info_t , sf_ble_prf_hid_report_desc_t
- dhcpServerStart : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- dhcpServerStop : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- diagnosis : ctsu_api_t
- digfilter : cac_ref_clock_config_t
- direct_addr_type : sf_ble_adv_info_t
- direct_bd_addr : sf_ble_adv_info_t
- disable : elc_api_t , external_irq_api_t , input_capture_api_t , keymatrix_api_t , transfer_api_t

- disableINFABTnotification : ptp_api_t
- disableTimer : ptp_api_t
- disc_mode : sf_ble_adv_info_t
- disc_time : sf_ble_cfg_t
- disconnect : sf_ble_api_t
- discovery_type : sf_ble_char_discovery_req_t , sf_ble_scan_info_t , sf_ble_service_discovery_req_t
- DISEL : dtc_reg_t
- disp_en : sf_ble_sm_tk_info_t
- display_cyc : display_timing_t
- display_list_flushed : sf_el_gx_instance_ctrl_t
- dithering_mode : glcd_cfg_t
- dithering_on : display_output_cfg_t
- dithering_pattern_A : glcd_cfg_t
- dithering_pattern_B : glcd_cfg_t
- dithering_pattern_C : glcd_cfg_t
- dithering_pattern_D : glcd_cfg_t
- divider : cac_meas_clock_config_t , cac_ref_clock_config_t , cgc_clock_cfg_t
- DM : dtc_reg_t
- dma_req_ipl : sdmmc_cfg_t
- do_run : sf_cellular_comms_extend_cfg_t
- dodir : doc_data_t
- dodsr : doc_data_t
- domain_params : sf_crypto_key_cfg_t , sf_crypto_key_instance_ctrl_t
- dri_marker : jpeg_encode_cfg_t
- drift_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- drive_volt_gen : slcdc_cfg_t
- driver_packets_queued : NX_REC
- driver_rx_bd : NX_REC
- driver_rx_bd_index : NX_REC
- driver_task_priority : sf_wifi_on_gt202_cfg_t
- driver_tx_bd : NX_REC
- driver_tx_bd_in_use : NX_REC
- driver_tx_bd_index : NX_REC
- driver_tx_packet_queue : NX_REC
- driver_tx_packet_queue_end : NX_REC
- driver_tx_release_index : NX_REC
- dst_offset : sf_ble_cts_local_time_t , sf_ble_prf_ndcs_time_dst_t
- dtc_state_in_snooze : lpmv2_mcu_cfg_t
- dtc_transfer_length : sf_adc_periodic_instance_ctrl_t
- dtim : sf_wifi_cfg_t
- DTS : dtc_reg_t
- dummy_read_completed : riic_instance_ctrl_t
- duplicate_filt : sf_ble_scan_info_t
- duty_cycle : timer_cfg_t
- duty_cycle_unit : timer_cfg_t
- dutyCycleSet : timer_api_t

- e -

- each_len : RBLE_GATT_INFO_DATA

- ebr : sf_el_fx_media_boot_record_table_info_t
- EcdhSharedSecretCompute : sf_crypto_key_api_t
- echo : sf_console_cfg_t , sf_console_instance_ctrl_t
- edge : cac_ref_clock_config_t , input_capture_cfg_t
- ediv : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- edmac_ptr : NX_REC
- elc_event : adc_info_t , timer_info_t
- elc_peripheral : adc_info_t
- elem_index : sf_touch_ctsu_button_cfg_t
- elmt : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t
- elmt_hdl : RBLE_GATT_RELIABLE_WRITE
- enable : display_brightness_t , display_contrast_t , elc_api_t , external_irq_api_t , gamma_correction_t , input_capture_api_t , keymatrix_api_t , transfer_api_t
- enable_charge_pump : dac8_extended_cfg_t , dac_extended_cfg_t
- enable_filter : gpt_input_capture_extend_t
- enable_level : gpt_input_capture_extend_t
- enableINFABTnotification : ptp_api_t
- encoded_lines : jpeg_encode_instance_ctrl_t
- encrypt : aes_api_t , rsa_api_t , tdes_api_t
- encryptFinal : aes_api_t
- encryption : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_scan_t
- end : sf_message_instance_range_t , sf_audio_playback_instance_ctrl_t
- end_handle : sf_ble_char_discovery_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- end_hdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_SET_PERM , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , RBLE_GATT_SVC_RANGE_LIST
- end_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- endian : display_output_cfg_t , pdc_cfg_t , pdc_instance_ctrl_t
- endian_flag : crypto_cfg_t
- energy_expended : sf_ble_hrp_api_hrmeas_t
- entry_len : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t
- eptpc_flag : ptp_instance_ctrl_t
- erase : flash_api_t , qspi_api_t , sdmmc_api_t , sf_memory_api_t
- erase_block_count : sf_el_lx_nor_callback_args_t
- erase_block_number : sf_el_lx_nor_callback_args_t
- erase_sector_count : sdmmc_info_t
- eri_ipl : i2c_cfg_t , spi_cfg_t , uart_cfg_t
- eri_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t
- err : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- err_irq : flash_hp_instance_ctrl_t
- err_irq_ipl : flash_cfg_t
- error : sf_crypto_callback_args_t , sf_crypto_hash_callback_args_t , sf_el_gx_callback_args_t
- error_adjustment_type : rtc_cfg_t
- error_adjustment_value : rtc_cfg_t
- error_check_index : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t
- error_code : jpeg_decode_instance_ctrl_t
- error_ipl : can_cfg_t
- error_irq : can_instance_ctrl_t
- errorAdjustmentModeSet : rtc_api_t
- errorAdjustmentSet : rtc_api_t
- ether_frame_type : ptpedmac_callback_args_t
- etherc_ptr : NX_REC

- event : adc_callback_args_t , cac_callback_args_t , can_callback_args_t , cgc_callback_args_t , ctsu_callback_args_t , display_callback_args_t , doc_callback_args_t , doc_cfg_t , doc_instance_ctrl_t , elc_link_t , flash_callback_args_t , i2c_callback_args_t , i2s_callback_args_t , input_capture_callback_args_t , pdc_callback_args_t , ptp_callback_args_t , ptpedmac_callback_args_t , rtc_callback_args_t , sdmmc_callback_args_t , sf_adc_periodic_callback_args_t , sf_audio_playback_hw_callback_args_t , sf_ble_event_info_t , sf_cellular_callback_args_t , sf_comms_callback_args_t , sf_crypto_callback_args_t , sf_el_gx_callback_args_t , sf_el_lx_nor_callback_args_t , sf_external_irq_cfg_t , sf_message_callback_args_t , sf_power_profiles_v2_callback_args_t , sf_wifi_callback_args_t , spi_callback_args_t , timer_callback_args_t , uart_callback_args_t
- event_class : sf_message_subscriber_list_t
- event_type : sf_ble_scan_t , sf_touch_panel_v2_payload_t
- eventflag : sf_block_media_sdmmc_instance_ctrl_t , sf_uart_comms_instance_ctrl_t
- eventInfoGet : fmi_api_t
- events : sf_jpeg_decode_instance_ctrl_t
- exe_wr_ena : RBLE_GATT_EXE_WR_CHAR_REQ
- expect_resp_size : RBLE_GATT_UUID_TYPE
- expected_result_size : sf_ble_char_multiple_read_req_t

- f -

- fade_control : display_layer_t
- fade_speed : display_layer_t
- fade_status : display_status_t
- fclk_div : cgc_system_clock_cfg_t
- ferr_interrupt_enabled : cac_cfg_t
- fifo_access_bytes : ssi_instance_ctrl_t
- fifo_addr : UX_HCD_SYNERGY_FIFO
- fifo_ctrl : UX_HCD_SYNERGY_FIFO
- fifo_depth : sci_uart_instance_ctrl_t
- fifo_mode : crc_cfg_t , crc_instance_ctrl_t
- fifo_num_stages : bsp_feature_ssi_t
- fifo_sel : UX_HCD_SYNERGY_FIFO
- filt_policy : sf_ble_scan_info_t
- filter : comparator_cfg_t
- filter_enable : external_irq_cfg_t
- filterDisable : external_irq_api_t
- filterEnable : external_irq_api_t
- first_coefficient : ctsu_correction_info_t
- first_val : ctsu_correction_info_t
- flag_stable_meas : sf_ble_blp_meas_info_t , sf_ble_prf_htp_temp_info_t
- flags : agt_input_capture_instance_ctrl_t , sf_audio_playback_common_instance_ctrl_t , sf_ble_blp_meas_info_t , sf_ble_hrp_api_hrmeas_t , sf_ble_prf_htp_temp_info_t , sf_touch_panel_v2_instance_ctrl_t
- flash_cf_macros : bsp_feature_flash_lp
- flash_clock_src : bsp_feature_flash_lp
- flush : sf_memory_api_t
- format : display_input_cfg_t , display_output_cfg_t
- format_status : sf_el_fx_media_partition_data_info_t
- fotaCheck : sf_cellular_api_t

- fotaStart : sf_cellular_api_t
- fotaStop : sf_cellular_api_t
- fractions256 : sf_ble_prf_cts_curr_time_t
- fragmentation : sf_wifi_cfg_t
- frame_end_ipl : pdc_cfg_t
- frame_end_irq : pdc_instance_ctrl_t
- frame_format : ptp_cfg_t , ptp_instance_ctrl_t
- frame_type : can_mailbox_t
- freq_hz_min : sf_spi_bus_t
- frequency_error_ipl : cac_cfg_t
- frequency_error_irq : cac_instance_ctrl_t
- fw_version : sf_cellular_info_t

- g -

- g : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- gain : gamma_correction_t
- gap_name : sf_ble_provisioning_t
- gap_role : sf_ble_provisioning_t
- gateway : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- gattAddCustomProfiles : sf_ble_api_t
- gattCharDescDiscovery : sf_ble_api_t
- gattCharDiscovery : sf_ble_api_t
- gattCharExecuteWrite : sf_ble_api_t
- gattCharRead : sf_ble_api_t
- gattCharWrite : sf_ble_api_t
- gattCharWriteLocal : sf_ble_api_t
- gattSendIndicate : sf_ble_api_t
- gattSendNotify : sf_ble_api_t
- gattServiceDiscovery : sf_ble_api_t
- gattWriteResponse : sf_ble_api_t
- generator_point : sf_crypto_key_cfg_t , sf_crypto_key_instance_ctrl_t
- getGcmTag : aes_api_t
- getLocalClock : ptp_api_t
- getMasterPortID : ptp_api_t
- getMessageReceptionConfig : ptp_api_t
- getSyncInfo : ptp_api_t
- getWorst10Values : ptp_api_t
- global_open : sf_el_fx_media_info_t
- group_b_sensors_allowed : bsp_feature_adc_t
- gtioca : timer_on_gpt_cfg_t
- gtioca_output_enabled : gpt_instance_ctrl_t
- gtiocb : timer_on_gpt_cfg_t
- gtiocb_output_enabled : gpt_instance_ctrl_t

- h -

- handle : sf_ble_char_read_by_uuid_rsp_t , sf_ble_char_read_req_t , sf_ble_char_write_req_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t
- handles : sf_ble_char_multiple_read_req_t
- handles_cnt : sf_ble_char_read_req_t
- has_bclk : bsp_feature_cgc_t
- has_card_detection : bsp_feature_sdhi_t
- has_chargepump : bsp_feature_dac_t
- has_davrefcr : bsp_feature_dac_t
- has_digital_filter : bsp_feature_lvd_t
- has_dssby : bsp_feature_lpmv2_t
- has_ethernet : bsp_feature_ioport_t
- has_fclk : bsp_feature_cgc_t
- has_ir_flag : bsp_feature_icu_t
- has_lcd_clock : bsp_feature_cgc_t
- has_pclka : bsp_feature_cgc_t
- has_pclkb : bsp_feature_cgc_t
- has_pclkc : bsp_feature_cgc_t
- has_pclkd : bsp_feature_cgc_t
- has_sample_hold_reg : bsp_feature_adc_t
- has_sdadc_clock : bsp_feature_cgc_t
- has_sdram_clock : bsp_feature_cgc_t
- has_subosc_speed : bsp_feature_cgc_t
- has_usb_clock_div : bsp_feature_cgc_t
- has_vbatt_pins : bsp_feature_ioport_t
- hash : hash_ctrl_t
- hash_context : sf_crypto_hash_instance_ctrl_t
- hash_type : sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t
- hashFinal : sf_crypto_hash_api_t
- hashInit : sf_crypto_hash_api_t
- hashSign : dsa_api_t
- hashUpdate : hash_api_t , sf_crypto_hash_api_t
- hashVerify : dsa_api_t
- hc : sdmmc_info_t
- header : sf_audio_playback_data_t
- heart_rate_measure : sf_ble_hrp_api_hrmeas_t
- help : sf_console_command_t
- high_speed_freq_hz : bsp_feature_cgc_t
- high_throughput : sf_wifi_cfg_t
- hoco_freq_hz : bsp_feature_cgc_t
- hoco_state : cgc_clocks_cfg_t
- horizontal_resolution : jpeg_encode_cfg_t , jpeg_encode_raw_image_parameters
- horizontal_stride : jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , jpeg_encode_raw_image_parameters
- horizontal_subsample : jpeg_decode_instance_ctrl_t
- horizontalStrideSet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- hour : sf_ble_prf_cts_date_time_t
- hour_match : rtc_alarm_time_t
- hours_since_update : sf_ble_cts_ref_time_t
- hs_timing : sdmmc_info_t
- hsize : display_input_cfg_t , glcd_ctrl_t
- hsize_pixels : sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- hstride : display_input_cfg_t
- hsync : pdc_state_t

- hsync_polarity : pdc_cfg_t , pdc_instance_ctrl_t
- htiming : display_output_cfg_t
- hw : sdmmc_cfg_t , sdmmc_instance_ctrl_t
- hw_cfg : rtc_cfg_t
- hw_mode : sf_wifi_cfg_t , sf_wifi_qca4010_cfg_t , sf_wifi_scan_t
- hysteresis : sf_touch_ctsu_button_cfg_t

- i -

- i2c_hw_err_event : i2c_callback_args_t
- iclk_div : bsp_feature_cgc_t , cgc_system_clock_cfg_t
- id : can_frame_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , sf_ble_blp_meas_info_t
- id_key : sf_ble_sec_info_t
- id_mode : can_cfg_t , can_instance_ctrl_t
- idCodeSet : flash_api_t
- idle_err_ipl : i2s_cfg_t
- ikey_dist : sf_ble_bonding_req_ind_t , sf_ble_bonding_start_t
- ikeys : sf_ble_bonding_response_t
- image_size : jpeg_encode_callback_args_t
- imageParameterSet : jpeg_encode_api_t
- imageSizeGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- imageSubsampleSet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- imei : sf_cellular_info_t
- imsi : sf_cellular_network_status_t
- in_progress : transfer_properties_t
- in_use : sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- incl : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t::incl_list_u
- index : sf_ble_bonding_req_ind_t , sf_wifi_qca4010_provisioning_t , sf_audio_playback_instance_ctrl_t
- indicateEvent : ptp_api_t
- infabt_flag : ptp_instance_ctrl_t
- info : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , slcdc_instance_ctrl_t
- info_transfer : pdc_instance_ctrl_t
- infoGet : adc_api_t , can_api_t , comparator_api_t , dac_api_t , flash_api_t , i2s_api_t , input_capture_api_t , opamp_api_t , qspi_api_t , rtc_api_t , sdmmc_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_cellular_api_t , sf_memory_api_t , sf_wifi_api_t , timer_api_t , transfer_api_t , uart_api_t
- inherit_frame_layer : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- init : analog_connect_api_t , cgc_api_t , elc_api_t , fmi_api_t , ioport_api_t , lpmv2_api_t
- init_done : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- init_filt_type : sf_ble_connection_t
- init_status : sf_el_fx_media_info_t
- input : display_cfg_t , display_runtime_cfg_t , opamp_trim_args_t , sdadc_channel_cfg_t , sf_console_instance_ctrl_t
- input_data_format : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- inputBufferSet : jpeg_decode_api_t , jpeg_encode_api_t , sf_jpeg_decode_api_t
- inputRegisterWrite : doc_api_t
- inst_idx : sf_ble_onboard_profile_cccd_changed_t , sf_ble_prf_hid_change_event_t , sf_ble_prf_hid_report_ind_t

- instance_range : sf_message_subscriber_t
- int_irq : ssi_instance_ctrl_t
- interfaceGet : crypto_api_t
- intv : st_sf_ble_prf_htp_meas_intv_val_t
- invert : comparator_cfg_t , sdadc_channel_cfg_t
- io_cap : sf_ble_bonding_req_ind_t , sf_ble_bonding_response_t
- io_port_state : lpmv2_mcu_cfg_t
- iocap : sf_ble_bonding_start_t
- ioctl : sf_block_media_api_t
- IoIntEnable : sdmmc_api_t
- ip_addr : sf_cellular_info_t , sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- ip_addr_mode : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- ip_ptr : NX_REC
- ipAddressCfg : sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_onchip_stack_api_t
- ir_flag_stat : dmac_instance_ctrl_t
- irq : agt_instance_ctrl_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , gpt_instance_ctrl_t , icu_instance_ctrl_t , kint_instance_ctrl_t , NX_REC , pdc_instance_ctrl_t , transfer_info_t
- irq_ipl : comparator_cfg_t , doc_cfg_t , external_irq_cfg_t , flash_cfg_t , keymatrix_cfg_t , pdc_cfg_t , ptp_cfg_t , ptpedmac_cfg_t , timer_cfg_t , transfer_cfg_t
- irqDisable : rtc_api_t
- irqEnable : rtc_api_t
- is_dac_ramped_up : sf_audio_playback_hw_dac_instance_ctrl_t
- is_data_mode_on : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- is_opened : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- is_signed : sf_audio_playback_data_type_t
- iwdt_open : iwdt_instance_ctrl_t

- j -

- jdti_ipl : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- jedi_ipl : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- jpegbuffer_size : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t

- k -

- key : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t
- key_code : sf_ble_sm_key_ind_t
- key_data_length : sf_crypto_signature_context_t
- key_format : key_installation_key_t , rsa_key_t
- key_size : key_installation_key_t , sf_ble_bonding_start_t , sf_ble_sec_enc_start_ind_t , sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_cfg_t , sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- key_type : sf_crypto_cipher_cfg_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_cfg_t ,

sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- keyCreate : ecc_api_t , rsa_api_t
- keyGenerate : sf_crypto_key_api_t
- keyInstall : key_installation_api_t , sf_crypto_key_installation_api_t
- keySet : arc4_api_t

- l -

- last_payload : sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- lastCaptureGet : input_capture_api_t
- layer : display_cfg_t , display_runtime_cfg_t
- layerChange : display_api_t
- lcdClockCfg : cgc_api_t
- lcdClockDisable : cgc_api_t
- lcdClockEnable : cgc_api_t
- le_scan_interval : sf_ble_prf_scps_scan_intv_t
- le_scan_window : sf_ble_prf_scps_scan_intv_t
- led_count : bsp_leds_t
- len : RBLE_GATT_INFO_DATA , RBLE_GATT_QUERY_RESULT , sf_ble_attr_info_t
- length : adc_info_t , arc4_cfg_t , hash_ctrl_t , rsa_key_t , sf_adc_periodic_instance_ctrl_t , sf_ble_gatt_attr_event_t , sf_cellular_callback_args_t , sf_wifi_callback_args_t , transfer_info_t
- line_descending_enable : display_input_cfg_t
- line_detect_ipl : display_cfg_t
- lines_repeat_enable : display_input_cfg_t
- lines_repeat_times : display_input_cfg_t
- lines_to_encoded : jpeg_encode_instance_ctrl_t
- linesDecodedGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- link_count : elc_cfg_t
- link_established : NX_REC
- link_list : elc_cfg_t
- link_quality : sf_wifi_info_t
- linkBreak : elc_api_t
- linkCheck : ptpedmac_api_t
- linkProcess : ptpedmac_api_t
- linkSet : elc_api_t
- list : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t::incl_list_u
- list_128 : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t
- listen : sf_ble_api_t

- loaded : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- local_advertise : phy_record_t
- local_ip : sf_cellular_nsal_cfg_t , sf_cellular_socket_info_t
- local_port : sf_cellular_socket_info_t
- local_time : sf_ble_prf_tip_write_data_t
- lock : bsp_lock_t , sf_comms_api_t , sf_crypto_api_t , sf_i2c_api_t , sf_spi_api_t
- locked : sf_spi_instance_ctrl_t
- lockWait : sf_i2c_api_t , sf_spi_api_t
- loco_state : cgc_clocks_cfg_t
- loop_timeout : sf_audio_playback_data_t
- loopback : spi_on_rspi_cfg_t
- low_power_mode : lpmv2_cfg_t
- low_speed_max_freq_hz : bsp_feature_cgc_t
- low_speed_pclk_div_min : bsp_feature_cgc_t
- low_voltage_max_freq_hz : bsp_feature_cgc_t
- low_voltage_pclk_div_min : bsp_feature_cgc_t
- lower_level : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- lower_lvl_cfg : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- lowPowerApply : sf_power_profiles_v2_api_t
- lowPowerCfg : lpmv2_api_t
- lowPowerModeEnter : lpmv2_api_t
- ltk_key : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- lvd_callback_args : lvd_instance_ctrl_t
- lvl : sb_ble_prf_ias_set_alert_t

- m -

- mac_addr : sf_wifi_callback_args_t , sf_wifi_cfg_t , sf_wifi_qca4010_status_t
- macAddressGet : sf_wifi_api_t
- macAddressSet : sf_wifi_api_t
- mailbox : can_callback_args_t
- mailbox_count : can_cfg_t , can_instance_ctrl_t
- mailbox_id : can_mailbox_t
- mailbox_rx_ipl : can_cfg_t
- mailbox_rx_irq : can_instance_ctrl_t
- mailbox_tx_ipl : can_cfg_t
- mailbox_tx_irq : can_instance_ctrl_t
- mailbox_type : can_mailbox_t
- main_osc_freq_hz : bsp_feature_cgc_t
- mainclock_drive : bsp_feature_cgc_t
- mainosc_state : cgc_clocks_cfg_t
- major : ssp_pack_version_t
- manufacturer_id : qspi_instance_ctrl_t
- masterReadSlaveWrite : i2c_api_slave_t
- masterWriteSlaveRead : i2c_api_slave_t
- max_backoffs : sf_cellular_qctlcatm1_socket_cfg_t
- max_clock_frequency : bsp_feature_sdhi_t
- max_clock_rate : sdmmc_info_t
- max_enc_size : sf_ble_bonding_req_ind_t
- max_eraseable_size : qspi_instance_ctrl_t
- max_key_size : sf_ble_bonding_response_t

- max_resolution : adc_instance_ctrl_t
- max_resp_length : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- max_rto : sf_cellular_qctlcatm1_socket_cfg_t
- max_slaves : sf_ble_cfg_t
- max_stations : sf_wifi_cfg_t
- maximum_count : sf_thread_monitor_counter_min_max_t , sf_thread_monitor_thread_counter_t
- mbr : sf_el_fx_media_boot_record_table_info_t
- mclock_only : bsp_feature_can_t
- md : ctsu_cfg_t , ctsu_instance_ctrl_t
- MD : dtc_reg_t
- mday_match : rtc_alarm_time_t
- meas_info : sf_ble_blp_meas_recv_data_t
- meas_sts : sf_ble_blp_meas_info_t
- measurement_clock : cac_instance_ctrl_t
- measurement_end_ipl : cac_cfg_t
- measurement_end_irq : cac_instance_ctrl_t
- measurements_info : sf_ble_hrp_api_meas_ntf_t
- media_info : sf_el_fx_instance_ctrl_t
- media_type : sdmmc_hw_t
- mei_interrupt_enabled : cac_cfg_t
- memory_capacity : qspi_instance_ctrl_t
- memory_end_address : sf_memory_region_info_t
- memory_free_sectors : sf_el_fx_media_info_t
- memory_pool_size : sf_crypto_cfg_t
- memory_start_address : sf_memory_region_info_t
- memory_total_sectors : sf_el_fx_media_info_t
- memory_type : qspi_instance_ctrl_t
- menu_name : sf_console_menu_t
- menu_prev : sf_console_menu_t
- message_bytes : sf_crypto_hash_context_t
- message_bytes_buffered : sf_crypto_hash_context_t
- message_format : sf_crypto_signature_rsa_specific_params_t
- message_mode : can_cfg_t , can_instance_ctrl_t
- mfg_name : sf_cellular_info_t
- middle_speed_max_freq_hz : bsp_feature_cgc_t
- min : sf_ble_prf_cts_date_time_t
- min_match : rtc_alarm_time_t
- min_program_size_bytes : qspi_info_t
- min_stabilization_wait_us : comparator_info_t , opamp_info_t
- min_wait_time_hs_us : bsp_feature_opamp_t
- min_wait_time_lp_us : bsp_feature_opamp_t
- min_wait_time_ms_us : bsp_feature_opamp_t
- min_wait_time_us : bsp_feature_acmphs_t
- minimum_count : sf_thread_monitor_counter_min_max_t , sf_thread_monitor_thread_counter_t
- minimum_erase_size : sf_memory_region_info_t
- minimum_write_size : sf_memory_region_info_t
- minor : ssp_pack_version_t
- mint_irq : ptp_instance_ctrl_t
- moco_state : cgc_clocks_cfg_t
- mode : adc_cfg_t , adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , comparator_cfg_t , gpt_input_capture_instance_ctrl_t , input_capture_cfg_t , opamp_on_opamp_cfg_t , sdadc_calibrate_args_t , sdadc_instance_ctrl_t , sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t ,

sf_wifi_qca4010_status_t , timer_cfg_t , transfer_info_t
- mode_fault : spi_cfg_t
- modify : slcdc_api_t
- modrv_mask : bsp_feature_cgc_t
- modrv_shift : bsp_feature_cgc_t
- mon_match : rtc_alarm_time_t
- monitor_1_hi_threshold : bsp_feature_lvd_t
- monitor_1_low_threshold : bsp_feature_lvd_t
- monitor_2_hi_threshold : bsp_feature_lvd_t
- monitor_2_low_threshold : bsp_feature_lvd_t
- monitor_ipl : lvd_cfg_t
- monitor_number : lvd_callback_args_t , lvd_cfg_t , lvd_instance_ctrl_t
- month : sf_ble_prf_cts_date_time_t
- mosi_idle : spi_on_rspi_cfg_t
- MRA : dtc_reg_t
- MRA_b : dtc_reg_t
- MRB : dtc_reg_t
- MRB_b : dtc_reg_t
- msgbuf : hash_ctrl_t
- multicastListAdd : sf_wifi_api_t
- multicastListDelete : sf_wifi_api_t
- multiple_partitions_status : sf_el_fx_media_partition_info_t
- multiplier : cgc_clock_cfg_t
- mute : i2s_api_t
- mutex : sf_adc_periodic_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_i2s_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_jpeg_decode_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t , sf_uart_comms_instance_ctrl_t

- n -

- nak_response : sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- nattempts : trng_cfg_t , trng_ctrl_t
- nb_entry : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_By_Uuid_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Char_Desc_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_Incl_Comp_t
- nb_resp : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_128_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_All_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Disc_Svc_By_Uuid_Comp_t
- nb_uuid : RBLE_GATT_READ_CHAR_REQ
- nb_writes : RBLE_GATT_WRITE_RELIABLE_REQ
- nbiot_band_selection : sf_cellular_qctlcatm1_extended_cfg_t
- negation_delay : lvd_extend_t
- negation_delay_clock : bsp_feature_lvd_t

- netmask : sf_wifi_onchip_stack_ip_cfg_t , sf_wifi_qca4010_onchip_stack_ip_cfg_t
- networkConnect : sf_cellular_api_t
- networkDisconnect : sf_cellular_api_t
- networkStatusGet : sf_cellular_api_t
- new_alert : sf_ble_anp_api_new_alert_ntf_t
- new_conn_fd : sf_cellular_socket_info_t
- new_line : sf_console_instance_ctrl_t
- next_dst : sf_ble_prf_tip_write_data_t
- next_length : sf_audio_playback_common_instance_ctrl_t
- noise_level : sf_wifi_info_t
- noisecancel_en : uart_on_sci_cfg_t
- nor_driver_initialize : sf_block_media_on_lx_nor_cfg_t
- notify_request : riic_slave_instance_ctrl_t
- num_address_bytes : qspi_instance_ctrl_t
- num_blocks : transfer_info_t
- num_buttons : sf_touch_ctsu_cfg_t
- num_commands : sf_console_menu_t
- num_elements : ctsu_instance_ctrl_t , sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- num_erase_sizes : qspi_info_t
- num_moving_average : ctsu_cfg_t , ctsu_instance_ctrl_t
- num_new_samples : sf_adc_periodic_callback_args_t
- num_pref_ops : sf_cellular_cfg_t
- num_regions : flash_fmi_regions_t
- num_rx : ctsu_cfg_t
- num_sliders : sf_touch_ctsu_cfg_t
- num_states : adc_sample_state_t
- num_tx : ctsu_cfg_t
- num_uarts : sf_wifi_qca4010_cfg_t , sf_wifi_qca4010_instance_cfg_t
- num_wheels : sf_touch_ctsu_cfg_t
- number : sf_touch_ctsu_cfg_t
- number_of_buffers : sf_message_instance_ctrl_t
- number_of_connections : analog_connect_table_t
- number_of_nodes : sf_message_subscriber_list_t
- number_of_pins : ioport_cfg_t
- number_of_regions : sf_memory_info_t
- number_of_subscriber_groups : sf_message_instance_ctrl_t
- nwscanseq : sf_cellular_qctlcatm1_extended_cfg_t
- nx_driver_phy_polling_requested : NX_REC
- nx_state : NX_REC

- o -

- off_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t
- offset : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_READ_CHAR_REQ , sf_ble_char_read_req_t , sf_ble_char_write_req_t , sf_ble_gatt_attr_event_t , sf_ble_write_cmd_event_data_t , sf_el_fx_media_partition_data_info_t
- offset_byte : dmac_instance_ctrl_t , transfer_on_dmac_cfg_t
- offsetSet : adc_api_t
- ok_check_index : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t
- on_freq : sf_touch_ctsu_button_info_t , sf_touch_ctsu_cfg_t

- onbpClientReadChar : sf_ble_onboard_profile_api_t
- onbpClientWriteCCCD : sf_ble_onboard_profile_api_t
- onbpClientWriteChar : sf_ble_onboard_profile_api_t
- onbpDisable : sf_ble_onboard_profile_api_t
- onbpEnable : sf_ble_onboard_profile_api_t
- onbpServerSendIndication : sf_ble_onboard_profile_api_t
- onbpServerSendNotification : sf_ble_onboard_profile_api_t
- onbpServerWriteData : sf_ble_onboard_profile_api_t
- one_shot : gpt_instance_ctrl_t
- oob_data_flg : sf_ble_bonding_req_ind_t
- op : sf_cellular_cfg_t
- op_name : sf_cellular_network_status_t , sf_cellular_op_t
- op_name_format : sf_cellular_op_t
- op_select_mode : sf_cellular_cfg_t
- open : adc_api_t , aes_api_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , arc4_api_t , arc4_ctrl_t , cac_api_t , can_api_t , can_instance_ctrl_t , comparator_api_t , crc_api_t , crc_instance_ctrl_t , crypto_api_t , ctsu_api_t , ctsu_instance_ctrl_t , dac_api_t , display_api_t , doc_api_t , doc_instance_ctrl_t , dsa_api_t , ecc_api_t , external_irq_api_t , flash_api_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , icu_instance_ctrl_t , input_capture_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , kint_instance_ctrl_t , lvd_api_t , opamp_api_t , pdc_api_t , pdc_instance_ctrl_t , ptp_api_t , ptp_instance_ctrl_t , ptpedmac_api_t , ptpedmac_instance_ctrl_t , qspi_api_t , qspi_instance_ctrl_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rsa_api_t , rtc_api_t , rtc_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_uart_instance_ctrl_t , sdmmc_api_t , sdmmc_instance_ctrl_t , sf_adc_periodic_api_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_api_t , sf_audio_playback_common_instance_ctrl_t , sf_audio_playback_hw_api_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_api_t , sf_audio_record_i2s_instance_ctrl_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_qspi_instance_ctrl_t , sf_block_media_ram_instance_ctrl_t , sf_block_media_sdmmc_instance_ctrl_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t , sf_el_fx_instance_ctrl_t , sf_el_gx_api_t , sf_el_lx_nor_instance_ctrl_t , sf_external_irq_api_t , sf_external_irq_instance_ctrl_t , sf_i2c_api_t , sf_jpeg_decode_api_t , sf_jpeg_decode_instance_ctrl_t , sf_memory_api_t , sf_memory_qspi_nor_instance_ctrl_t , sf_message_api_t , sf_power_profiles_v2_api_t , sf_power_profiles_v2_ctrl_t , sf_socket_api_t , sf_spi_api_t , sf_thread_monitor_api_t , sf_thread_monitor_instance_ctrl_t , sf_touch_ctsu_api_t , sf_touch_ctsu_instance_ctrl_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_touch_panel_v2_instance_ctrl_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , ssi_instance_ctrl_t , sf_audio_playback_instance_ctrl_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t , wdt_api_t
- open_counter : sf_crypto_instance_ctrl_t
- open_status : sf_el_fx_media_partition_data_info_t
- opened : adc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , lvd_instance_ctrl_t , opamp_instance_ctrl_t , sdadc_instance_ctrl_t
- operating_channel_mask : opamp_status_t
- operating_mode : i2s_cfg_t , spi_cfg_t
- operation_context : sf_crypto_signature_instance_ctrl_t
- operation_mode : can_instance_ctrl_t , sf_crypto_signature_context_t
- operation_state : sf_crypto_signature_instance_ctrl_t
- operator_code : sf_cellular_network_status_t

- oscStopDetect : cgc_api_t
- oscStopStatusClear : cgc_api_t
- outbuffer_size : jpeg_decode_instance_ctrl_t
- output : display_cfg_t
- output_amplifier_enabled : dac_cfg_t
- output_buffer_size : jpeg_encode_instance_ctrl_t
- output_data_format : jpeg_decode_cfg_t , jpeg_encode_cfg_t
- output_enabled : gpt_output_pin_t
- output_inverted : timer_on_agt_cfg_t
- output_port_enable : lpmv2_mcu_cfg_t
- outputBufferSet : jpeg_decode_api_t , jpeg_encode_api_t , sf_jpeg_decode_api_t
- outputEnable : comparator_api_t
- over_current : adc_cfg_t , adc_instance_ctrl_t
- overflow_ipl : cac_cfg_t
- overflow_irq : agt_input_capture_instance_ctrl_t , cac_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- overflow_irq_ipl : input_capture_cfg_t
- overflows : input_capture_callback_args_t , input_capture_capture_t
- overflows_current : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t
- overflows_last : gpt_input_capture_instance_ctrl_t
- oversampling : sdadc_channel_cfg_t
- ovf_interrupt_enabled : cac_cfg_t
- own_addr_type : sf_ble_adv_info_t , sf_ble_cfg_t

- p -

- p_address : adc_info_t
- p_aglorithm_specific_params : sf_crypto_signature_context_t
- p_api : adc_instance_t , aes_instance_t , analog_connect_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , cgc_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t , dsa_instance_t , ecc_instance_t , elc_instance_t , external_irq_instance_t , flash_instance_t , fmi_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , ioport_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lpmv2_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t , sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t ,

trng_instance_t , uart_instance_t , wdt_instance_t
- p_app_ptp_rx_desc : ptpedmac_instance_ctrl_t
- p_attr_value : sf_ble_char_attribute_t , sf_ble_svc_attribute_t
- p_auth_tag : sf_crypto_cipher_aes_init_params_t
- p_base : display_clut_cfg_t , display_input_cfg_t
- p_bit_timing : can_cfg_t
- p_ble_callback : sf_ble_provisioning_t
- p_block_array : flash_fmi_regions_t
- p_block_pool_name : sf_message_cfg_t
- p_buff : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , sf_cellular_cmd_resp_t , sf_wifi_qca4010_cmd_resp_t
- p_buffer : pdc_callback_args_t , pdc_cfg_t
- p_buffer_pool_rx : sf_wifi_cfg_t
- p_bus : sf_i2c_cfg_t , sf_i2c_instance_ctrl_t , sf_spi_cfg_t , sf_spi_instance_ctrl_t
- p_bus_name : sf_i2c_bus_t , sf_spi_bus_t
- p_buttons : sf_touch_ctsu_cfg_t
- p_callback : adc_cfg_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_cfg_t , cac_instance_ctrl_t , can_cfg_t , can_instance_ctrl_t , comparator_cfg_t , ctsu_cfg_t , ctsu_instance_ctrl_t , display_cfg_t , dmac_instance_ctrl_t , doc_cfg_t , doc_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_cfg_t , flash_cfg_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_cfg_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_cfg_t , iwdt_instance_ctrl_t , jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , keymatrix_cfg_t , lvd_cfg_t , lvd_instance_ctrl_t , pdc_cfg_t , pdc_instance_ctrl_t , ptp_cfg_t , ptp_instance_ctrl_t , ptpedmac_cfg_t , ptpedmac_instance_ctrl_t , rspi_instance_ctrl_t , rtc_cfg_t , rtc_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_cfg_t , sf_audio_playback_hw_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_el_fx_config_t , sf_el_fx_instance_ctrl_t , sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t , sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t , sf_message_buffer_ctrl_t , sf_message_post_cfg_t , sf_power_profiles_v2_low_power_cfg_t , sf_wifi_cfg_t , sf_wifi_provisioning_t , sf_wifi_wps_t , spi_cfg_t , ssi_instance_ctrl_t , sf_audio_playback_instance_ctrl_t , timer_cfg_t , transfer_cfg_t , uart_cfg_t , wdt_cfg_t , wdt_instance_ctrl_t
- p_callback_memory : ctsu_instance_ctrl_t
- p_callback_rec : NX_REC
- p_canvas : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_capture_data_buffer : sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t
- p_cellular_mutex : sf_cellular_instance_cfg_t
- p_cfg : adc_instance_t , aes_instance_t , analog_connect_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , cgc_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t , dsa_instance_t , ecc_instance_t , elc_instance_t , external_irq_instance_t , flash_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , ioport_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lpmv2_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_cfg_t ,

sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t , sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_cfg_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t , trng_instance_t , uart_instance_t , wdt_instance_t

- p_channel_cfg : adc_instance_t
- p_channel_cfgs : adc_on_sdadc_cfg_t
- p_chap_get_challenge_cb : sf_cellular_nsal_cfg_t
- p_chap_get_responder_cb : sf_cellular_nsal_cfg_t
- p_chap_get_verify_cb : sf_cellular_nsal_cfg_t
- p_char_multiple_read_rsp : sf_ble_char_read_rsp_t
- p_char_read_by_handle_rsp : sf_ble_char_read_rsp_t
- p_char_read_by_uuid_rsp : sf_ble_char_read_rsp_t
- p_chip : sf_touch_panel_v2_instance_ctrl_t
- p_chipset : sf_wifi_info_t
- p_cipher_context_buffer : sf_crypto_cipher_instance_ctrl_t
- p_circular_queue : sf_cellular_circular_queue_cfg_t
- p_circular_queue_buffer : sf_cellular_circular_queue_cfg_t
- p_circular_queue_cfg : sf_cellular_extended_cfg_t
- p_clock_cfg : sf_power_profiles_v2_run_cfg_t
- p_clut : display_clut_t
- p_cmd : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- p_cmd_param_callback : sf_cellular_cfg_t
- p_cmd_queue_ptr : sf_wifi_qca4010_queue_cfg_t
- p_cmd_set : sf_cellular_cfg_t , sf_wifi_qca4010_cfg_t
- p_common_cfg : sf_audio_playback_cfg_t
- p_common_ctrl : sf_audio_playback_cfg_t , sf_audio_playback_instance_ctrl_t
- p_comms : sf_console_cfg_t , sf_console_instance_ctrl_t
- p_config : sf_el_fx_t
- p_connection_table : analog_connect_table_t
- p_context : adc_callback_args_t , adc_cfg_t , adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_callback_args_t , cac_cfg_t , cac_instance_ctrl_t , can_callback_args_t , can_cfg_t , can_instance_ctrl_t , cgc_callback_args_t , comparator_callback_args_t , comparator_cfg_t , ctsu_callback_args_t , ctsu_cfg_t , ctsu_instance_ctrl_t , display_callback_args_t , display_cfg_t , dmac_instance_ctrl_t , doc_callback_args_t , doc_cfg_t , doc_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_callback_args_t , external_irq_cfg_t , flash_callback_args_t , flash_cfg_t , glcd_ctrl_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , i2c_callback_args_t , i2c_cfg_t , i2s_callback_args_t , i2s_cfg_t , icu_instance_ctrl_t , input_capture_callback_args_t , input_capture_cfg_t , iwdt_instance_ctrl_t , jpeg_decode_callback_args_t , jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_callback_args_t , jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , keymatrix_callback_args_t , keymatrix_cfg_t , lvd_callback_args_t , lvd_cfg_t , pdc_callback_args_t , pdc_cfg_t , pdc_instance_ctrl_t , ptp_callback_args_t , ptp_cfg_t , ptp_instance_ctrl_t , ptpedmac_callback_args_t , ptpedmac_cfg_t , ptpedmac_instance_ctrl_t , rspi_instance_ctrl_t , rtc_callback_args_t , rtc_cfg_t , rtc_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_callback_args_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t ,

sf_adc_periodic_callback_args_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_callback_args_t , sf_audio_playback_hw_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t , sf_audio_record_i2s_instance_ctrl_t , sf_cellular_callback_args_t , sf_cellular_cfg_t , sf_crypto_cfg_t , sf_crypto_instance_ctrl_t , sf_el_fx_callback_args_t , sf_el_fx_config_t , sf_el_fx_instance_ctrl_t , sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t , sf_el_lx_nor_callback_args_t , sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t , sf_message_buffer_ctrl_t , sf_message_callback_args_t , sf_message_post_cfg_t , sf_power_profiles_v2_callback_args_t , sf_power_profiles_v2_low_power_cfg_t , sf_touch_ctsu_cfg_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t , sf_touchpanel_v2_callback_args_t , sf_wifi_callback_args_t , sf_wifi_cfg_t , sf_wifi_qca4010_cfg_t , slcdc_instance_ctrl_t , spi_callback_args_t , spi_cfg_t , ssi_instance_ctrl_t , timer_callback_args_t , timer_cfg_t , transfer_callback_args_t , transfer_cfg_t , uart_callback_args_t , uart_cfg_t , wdt_callback_args_t , wdt_cfg_t , wdt_instance_ctrl_t

- p_correction_info : ctsu_instance_ctrl_t
- p_crypto_api : aes_cfg_t , aes_ctrl_t , arc4_cfg_t , arc4_ctrl_t , dsa_cfg_t , dsa_ctrl_t , ecc_cfg_t , ecc_ctrl_t , hash_cfg_t , hash_ctrl_t , key_installation_instance_ctrl_t , rsa_cfg_t , rsa_ctrl_t , tdes_cfg_t , tdes_ctrl_t , trng_cfg_t , trng_ctrl_t
- p_crypto_ctrl : aes_ctrl_t , arc4_ctrl_t , dsa_ctrl_t , ecc_ctrl_t , hash_cfg_t , key_installation_instance_ctrl_t , rsa_ctrl_t , trng_ctrl_t
- p_ctrl : adc_instance_t , aes_instance_t , arc4_instance_t , cac_instance_t , can_instance_t , comparator_instance_t , crc_instance_t , crypto_instance_t , ctsu_instance_t , dac_instance_t , display_instance_t , doc_instance_t , dsa_instance_t , ecc_instance_t , external_irq_instance_t , flash_instance_t , hash_instance_t , i2c_master_instance_t , i2c_slave_instance_t , i2s_instance_t , input_capture_instance_t , jpeg_decode_instance_t , jpeg_encode_instance_t , key_installation_instance_t , keymatrix_instance_t , lvd_instance_t , opamp_instance_t , pdc_instance_t , ptp_instance_t , ptpedmac_instance_t , qspi_instance_t , rsa_instance_t , rtc_instance_t , sdmmc_instance_t , sf_adc_periodic_instance_t , sf_audio_playback_hw_instance_t , sf_audio_playback_instance_t , sf_audio_record_instance_t , sf_ble_instance_t , sf_ble_onboard_profile_instance_t , sf_block_media_instance_t , sf_cellular_instance_t , sf_cellular_socket_instance_t , sf_comms_instance_t , sf_console_callback_args_t , sf_console_instance_t , sf_crypto_cipher_instance_t , sf_crypto_hash_instance_t , sf_crypto_instance_t , sf_crypto_key_installation_instance_t , sf_crypto_key_instance_t , sf_crypto_signature_instance_t , sf_crypto_trng_instance_t , sf_el_fx_t , sf_el_gx_instance_t , sf_external_irq_instance_t , sf_i2c_instance_t , sf_jpeg_decode_instance_t , sf_memory_instance_t , sf_message_instance_t , sf_power_profiles_v2_instance_t , sf_socket_instance_t , sf_spi_instance_t , sf_thread_monitor_instance_t , sf_touch_ctsu_instance_t , sf_touch_panel_chip_instance_t , sf_touch_panel_v2_instance_t , sf_wifi_instance_t , sf_wifi_onchip_stack_instance_t , sf_wifi_qca4010_instance_t , sf_wifi_qca4010_onchip_stack_instance_t , sf_wifi_qca4010_socket_instance_t , slcdc_instance_t , spi_instance_t , tdes_instance_t , timer_instance_t , transfer_instance_t , trng_instance_t , uart_instance_t , wdt_instance_t
- p_ctsu_cfg : ctsu_instance_ctrl_t
- p_ctsu_instance : sf_touch_ctsu_cfg_t , sf_touch_ctsu_instance_ctrl_t
- p_ctsuwr : ctsu_instance_ctrl_t
- p_current_buffer : pdc_instance_ctrl_t
- p_current_menu : sf_console_instance_ctrl_t
- p_data : key_installation_key_t , r_crypto_data_handle_t , rsa_key_t , sf_audio_playback_data_t , sf_ble_char_multiple_read_rsp_t , sf_ble_char_read_by_handle_rsp_t , sf_ble_char_read_by_uuid_rsp_t , sf_ble_char_write_req_t , sf_ble_event_info_t , sf_ble_gatt_notif_ind_event_data_t , sf_ble_write_cmd_event_data_t , sf_cellular_callback_args_t , sf_crypto_data_handle_t , sf_el_fx_media_partition_info_t ,

sf_wifi_callback_args_t , sf_audio_playback_instance_ctrl_t
- p_data_buffer : sf_adc_periodic_callback_args_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- p_delay_callback : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_delay_callback_context : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_dest : transfer_info_t
- p_disconnect_callback : sf_comms_telnet_cfg_t , sf_comms_telnet_instance_ctrl_t
- p_display : sf_el_gx_instance_ctrl_t
- p_display_instance : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_display_runtime_cfg : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_drift_buf : sf_touch_ctsu_button_info_t
- p_drift_count : sf_touch_ctsu_button_info_t
- p_driver_handle : sf_ble_ctrl_t , sf_cellular_ctrl_t , sf_wifi_ctrl_t , sf_wifi_qca4010_ctrl_t
- p_elem_index : sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- p_elements : ctsu_cfg_t
- p_erase_sizes_bytes : qspi_info_t
- p_eventflag : sf_wifi_qca4010_extended_cfg_t
- p_extend : adc_cfg_t , cac_cfg_t , can_cfg_t , comparator_cfg_t , crc_cfg_t , ctsu_cfg_t , display_cfg_t , external_irq_cfg_t , flash_cfg_t , i2c_cfg_t , i2s_cfg_t , input_capture_cfg_t , jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , key_installation_cfg_t , keymatrix_cfg_t , lpmv2_cfg_t , lvd_cfg_t , opamp_cfg_t , pdc_cfg_t , ptp_cfg_t , ptp_instance_ctrl_t , qspi_cfg_t , rtc_cfg_t , sdmmc_cfg_t , sf_adc_periodic_cfg_t , sf_audio_playback_common_cfg_t , sf_audio_playback_hw_cfg_t , sf_audio_record_cfg_t , sf_ble_cfg_t , sf_ble_onboard_profile_cfg_t , sf_block_media_cfg_t , sf_cellular_cfg_t , sf_cellular_nsal_cfg_t , sf_cellular_socket_cfg_t , sf_comms_cfg_t , sf_crypto_cfg_t , sf_crypto_cipher_cfg_t , sf_crypto_hash_cfg_t , sf_crypto_key_cfg_t , sf_crypto_key_installation_cfg_t , sf_crypto_signature_cfg_t , sf_crypto_trng_cfg_t , sf_el_fx_config_t , sf_memory_cfg_t , sf_power_profiles_v2_cfg_t , sf_power_profiles_v2_low_power_cfg_t , sf_power_profiles_v2_run_cfg_t , sf_socket_cfg_t , sf_thread_monitor_instance_ctrl_t , sf_touch_ctsu_cfg_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_cfg_t , sf_wifi_cfg_t , sf_wifi_onchip_stack_cfg_t , sf_wifi_qca4010_cfg_t , sf_wifi_qca4010_socket_cfg_t , spi_cfg_t , timer_cfg_t , transfer_cfg_t , uart_cfg_t , wdt_cfg_t
- p_extpin_ctrl : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- p_framebuffer_a : sf_el_gx_cfg_t
- p_framebuffer_b : sf_el_gx_cfg_t
- p_framebuffer_read : sf_el_gx_instance_ctrl_t
- p_framebuffer_write : sf_el_gx_instance_ctrl_t
- p_fwk_common_api : sf_crypto_key_instance_ctrl_t
- p_fwk_common_ctrl : sf_crypto_key_instance_ctrl_t
- p_gamma_correction : display_output_cfg_t
- p_hal_api : sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_hal_ctrl : sf_crypto_cipher_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_hidden_sector : sf_el_fx_callback_args_t
- p_huffman_croma_ac_table : jpeg_encode_cfg_t
- p_huffman_croma_dc_table : jpeg_encode_cfg_t
- p_huffman_luma_ac_table : jpeg_encode_cfg_t
- p_huffman_luma_dc_table : jpeg_encode_cfg_t
- p_hysteresis : sf_touch_ctsu_button_info_t
- p_info : transfer_cfg_t
- p_initial_menu : sf_console_cfg_t
- p_interface : sf_wifi_nsal_callback_args_t
- p_ioport_pin_tbl : sf_power_profiles_v2_run_cfg_t
- p_ioport_pin_tbl_enter : sf_power_profiles_v2_low_power_cfg_t

- p_ioport_pin_tbl_exit : sf_power_profiles_v2_low_power_cfg_t
- p_ip : sf_cellular_nsal_cfg_t , sf_wifi_nsal_callback_args_t
- p_iv : sf_crypto_cipher_aes_init_params_t
- p_jpegbuffer : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_key : arc4_cfg_t
- p_key_data : sf_crypto_signature_context_t
- p_leds : bsp_leds_t
- p_link_down_cb : sf_cellular_nsal_cfg_t
- p_link_up_cb : sf_cellular_nsal_cfg_t
- p_lock_mutex : sf_i2c_bus_t , sf_spi_bus_t
- p_low_lvl_ble : sf_ble_onboard_profile_cfg_t , sf_ble_onboard_profile_ctrl_t
- p_low_lvl_sf_comms : sf_ble_on_rl78g1d_cfg_t
- p_low_lvl_timer : sf_ble_on_rl78g1d_cfg_t
- p_lower_lvl : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_lower_lvl_adc : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t
- p_lower_lvl_adc_periodic : sf_audio_record_adc_instance_ctrl_t
- p_lower_lvl_api : sf_i2c_bus_t , sf_spi_bus_t
- p_lower_lvl_block_media : sf_el_fx_config_t , sf_el_fx_instance_ctrl_t
- p_lower_lvl_cellular : sf_cellular_socket_cfg_t , sf_cellular_socket_ctrl_t
- p_lower_lvl_cfg : sf_i2c_cfg_t , sf_spi_cfg_t
- p_lower_lvl_common : sf_crypto_key_installation_cfg_t , sf_crypto_trng_cfg_t
- p_lower_lvl_common_api : sf_crypto_key_installation_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_common_ctrl : sf_crypto_key_installation_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_crypto : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- p_lower_lvl_crypto_api : key_installation_cfg_t
- p_lower_lvl_crypto_common : sf_crypto_cipher_cfg_t , sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_key_cfg_t , sf_crypto_signature_cfg_t
- p_lower_lvl_crypto_trng : sf_crypto_cipher_cfg_t
- p_lower_lvl_ctrl : sf_i2c_instance_ctrl_t , sf_spi_instance_ctrl_t
- p_lower_lvl_dac : sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_framewrk : sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- p_lower_lvl_fwk_common_api : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_fwk_common_ctrl : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_hw : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t
- p_lower_lvl_i2c : sf_i2c_instance_ctrl_t
- p_lower_lvl_i2s : sf_audio_playback_hw_i2s_cfg_t , sf_audio_playback_hw_i2s_instance_ctrl_t , sf_audio_record_i2s_instance_ctrl_t
- p_lower_lvl_icu : sf_wifi_on_gt202_cfg_t
- p_lower_lvl_instance : sf_crypto_hash_cfg_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_key_installation_cfg_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_trng_cfg_t
- p_lower_lvl_irq : sf_external_irq_cfg_t , sf_external_irq_instance_ctrl_t , sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- p_lower_lvl_jpeg_decode : sf_jpeg_decode_cfg_t , sf_jpeg_decode_instance_ctrl_t
- p_lower_lvl_lpm : sf_power_profiles_v2_low_power_cfg_t
- p_lower_lvl_onchip_wifi : sf_socket_cfg_t , sf_socket_ctrl_t
- p_lower_lvl_onchip_wifi_qca4010 : sf_wifi_qca4010_socket_cfg_t , sf_wifi_qca4010_socket_ctrl_t

- p_lower_lvl_qspi : sf_block_media_qspi_instance_ctrl_t
- p_lower_lvl_sdmmc : sf_block_media_sdmmc_instance_ctrl_t
- p_lower_lvl_sf_crypto_hash : sf_crypto_signature_cfg_t , sf_crypto_signature_instance_ctrl_t
- p_lower_lvl_sf_crypto_trng_ctrl : sf_crypto_cipher_instance_ctrl_t
- p_lower_lvl_spi : sf_wifi_on_gt202_cfg_t
- p_lower_lvl_timer : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_transfer : pdc_cfg_t , pdc_instance_ctrl_t , sdmmc_cfg_t , sdmmc_instance_ctrl_t , sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_playback_hw_dac_cfg_t , sf_audio_playback_hw_dac_instance_ctrl_t
- p_lower_lvl_uart : sf_uart_comms_cfg_t , sf_uart_comms_instance_ctrl_t
- p_lower_lvl_wdt : sf_thread_monitor_cfg_t , sf_thread_monitor_instance_ctrl_t
- p_lower_lvl_wifi : sf_wifi_onchip_stack_cfg_t , sf_wifi_onchip_stack_ctrl_t
- p_lower_lvl_wifi_qca4010 : sf_wifi_qca4010_onchip_stack_cfg_t , sf_wifi_qca4010_onchip_stack_ctrl_t
- p_lx_nor_flash : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_mailbox : can_cfg_t , can_instance_ctrl_t
- p_mailbox_mask : can_extended_cfg_t
- p_memory_pool : sf_crypto_cfg_t
- p_memory_settings : sf_el_lx_nor_instance_cfg_t , sf_el_lx_nor_instance_ctrl_t
- p_message : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t
- p_message_buffer : sf_crypto_hash_context_t
- p_message_buffer_org : sf_crypto_hash_context_t
- p_message_digest : sf_crypto_hash_context_t
- p_message_digest_org : sf_crypto_hash_context_t
- p_modifiable_cmd_set : sf_cellular_cfg_t
- p_module_extended_cfg : sf_cellular_extended_cfg_t , sf_wifi_qca4010_extended_cfg_t
- p_mul_read_req : sf_ble_char_read_req_t
- p_mutual_pri_data : ctsu_instance_ctrl_t
- p_mutual_raw : ctsu_instance_ctrl_t
- p_mutual_snd_data : ctsu_instance_ctrl_t
- p_next_buffer : sf_audio_playback_common_instance_ctrl_t
- p_nor_flash : sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t
- p_nor_flash_name : sf_block_media_lx_nor_instance_ctrl_t , sf_block_media_on_lx_nor_cfg_t
- p_off_count : sf_touch_ctsu_button_info_t
- p_on_count : sf_touch_ctsu_button_info_t
- p_owner : sf_audio_playback_instance_ctrl_t
- p_pap_generate_login : sf_cellular_nsal_cfg_t
- p_pap_verify_login : sf_cellular_nsal_cfg_t
- p_pin_cfg_data : ioport_cfg_t
- p_position : sf_touch_ctsu_slider_info_t , sf_touch_ctsu_wheel_info_t
- p_ppp : sf_cellular_nsal_cfg_t
- p_ppp_invalid_packet_cb : sf_cellular_nsal_cfg_t
- p_ppp_packet_pool : sf_cellular_nsal_cfg_t
- p_ppp_send_byte : sf_cellular_nsal_cfg_t
- p_ppp_stack : sf_cellular_nsal_cfg_t
- p_prov_callback : sf_cellular_cfg_t
- p_ptpedmac_buffer : ptpedmac_instance_ctrl_t
- p_puk_pin : sf_cellular_cfg_t
- p_qspi : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- p_quant_croma_table : jpeg_encode_cfg_t
- p_quant_luma_table : jpeg_encode_cfg_t
- p_queue : sf_audio_playback_common_cfg_t , sf_audio_playback_common_instance_ctrl_t , sf_message_post_err_t , sf_message_subscriber_t
- p_queue_buffer : sf_wifi_qca4010_queue_cfg_t

- p_queue_cfg : sf_wifi_qca4010_extended_cfg_t
- p_r_uart_instance : sf_wifi_qca4010_uart_extend_cfg_t
- p_ram_buffer : sf_block_media_ram_instance_ctrl_t
- p_read_sim_pin_info_callback : sf_cellular_cfg_t
- p_recv_callback : sf_cellular_cfg_t
- p_reference : sf_touch_ctsu_button_info_t
- p_reg : adc_instance_ctrl_t , agt_input_capture_instance_ctrl_t , agt_instance_ctrl_t , cac_instance_ctrl_t , can_instance_ctrl_t , crc_instance_ctrl_t , ctsu_instance_ctrl_t , dac8_instance_ctrl_t , dac_instance_ctrl_t , dmac_instance_ctrl_t , doc_instance_ctrl_t , flash_hp_instance_ctrl_t , flash_lp_instance_ctrl_t , glcd_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , icu_instance_ctrl_t , iwdt_instance_ctrl_t , jpeg_decode_instance_ctrl_t , jpeg_encode_instance_ctrl_t , kint_instance_ctrl_t , lvd_instance_ctrl_t , opamp_instance_ctrl_t , pdc_instance_ctrl_t , ptp_instance_ctrl_t , ptpedmac_instance_ctrl_t , qspi_instance_ctrl_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , rtc_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , sdadc_instance_ctrl_t , sdmmc_instance_ctrl_t , slcdc_instance_ctrl_t , ssi_instance_ctrl_t , wdt_instance_ctrl_t
- p_reg_cfg : ptp_instance_ctrl_t
- p_reg_gen : ptp_instance_ctrl_t
- p_region_info : sf_el_lx_nor_instance_ctrl_t
- p_regions_info : sf_memory_info_t
- p_remaining_string : sf_console_callback_args_t
- p_resp_buff : sf_wifi_qca4010_instance_cfg_t
- p_rx_dest : ssi_instance_ctrl_t
- p_rx_dst : sci_uart_instance_ctrl_t
- p_sce_api_interfaces : crypto_cfg_t
- p_sce_long_plg_end_callback : crypto_cfg_t
- p_self_data : ctsu_instance_ctrl_t
- p_self_raw : ctsu_instance_ctrl_t
- p_service : sf_ble_char_attribute_t
- p_sf_comms_cfg : sf_cellular_extended_cfg_t
- p_sf_comms_instance : sf_cellular_comms_extend_cfg_t
- p_sf_comms_rx_thread : sf_cellular_comms_extend_cfg_t
- p_sf_comms_rx_thread_stack : sf_cellular_comms_extend_cfg_t
- p_sf_crypto_trng_api : sf_crypto_cipher_instance_ctrl_t
- p_sf_jpeg_decode_instance : sf_el_gx_cfg_t , sf_el_gx_instance_ctrl_t
- p_sim_pin : sf_cellular_cfg_t , sf_cellular_sim_pin_info_t
- p_sim_puk : sf_cellular_sim_pin_info_t
- p_sliders : sf_touch_ctsu_cfg_t
- p_socket_status_buffer : sf_cellular_instance_cfg_t
- p_src : transfer_info_t
- p_src_transfer : sf_adc_periodic_instance_ctrl_t
- p_stream : sf_audio_playback_common_instance_ctrl_t
- p_success_resp : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- p_sync_eventflag : sf_i2c_bus_t , sf_spi_bus_t
- p_thread : sf_thread_monitor_thread_counter_t
- p_threshold : sf_touch_ctsu_button_info_t , sf_touch_ctsu_slider_info_t , sf_touch_ctsu_wheel_info_t
- p_timer : i2s_cfg_t , ssi_instance_ctrl_t
- p_touch_cfg : sf_touch_ctsu_instance_ctrl_t
- p_transfer_data : sdmmc_instance_ctrl_t
- p_transfer_rx : ctsu_cfg_t , i2c_cfg_t , i2s_cfg_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , spi_cfg_t , ssi_instance_ctrl_t , uart_cfg_t
- p_transfer_tx : ctsu_cfg_t , i2c_cfg_t , i2s_cfg_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , spi_cfg_t , ssi_instance_ctrl_t , uart_cfg_t

- p_tsn_calib_regs : adc_instance_ctrl_t
- p_tsn_ctrl_regs : adc_instance_ctrl_t
- p_tuning_count : ctsu_instance_ctrl_t
- p_tuning_diff : ctsu_instance_ctrl_t
- p_tx_packet_buffer : sf_wifi_nsal_cfg_t
- p_tx_src : sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- p_uart_instance : sf_touch_ctsu_cfg_t
- p_uart_instance_objects : sf_wifi_qca4010_instance_cfg_t
- p_uart_instances : sf_wifi_qca4010_cfg_t
- p_value : sf_ble_gatt_attr_event_t
- p_wheels : sf_touch_ctsu_cfg_t
- p_wifi_mutex : sf_wifi_qca4010_instance_cfg_t
- p_wifi_nsal_cfg : sf_wifi_nsal_callback_args_t
- p_work_memory_start : sf_message_cfg_t
- page_size : qspi_instance_ctrl_t
- pageProgram : qspi_api_t
- parent_svc_handle : sf_ble_svc_attribute_t
- parity : spi_on_rspi_cfg_t , uart_cfg_t
- parse : sf_console_api_t
- partition : sf_el_fx_media_info_t
- password : sf_cellular_provisioning_t
- patch : ssp_pack_version_t
- pause : sf_audio_playback_api_t
- payload : sf_touch_panel_v2_instance_ctrl_t , sf_touchpanel_v2_callback_args_t
- payload_buffer : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- payload_length : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- payloadGet : sf_touch_panel_chip_api_t
- pckt_size : sf_cellular_socket_info_t
- pclk_div : external_irq_cfg_t
- pclka_div : cgc_system_clock_cfg_t
- pclkb_div : cgc_system_clock_cfg_t
- pclkc_div : cgc_system_clock_cfg_t
- pclkd_div : cgc_system_clock_cfg_t
- pcm_width : i2s_cfg_t
- pdp_type : sf_cellular_provisioning_t
- peer_addr : sf_ble_connect_info_t
- peer_addr_type : sf_ble_connect_info_t
- peer_ip : sf_cellular_nsal_cfg_t
- pend : sf_message_api_t
- pending_operation : sf_memory_qspi_nor_instance_ctrl_t
- pending_operation_size : sf_memory_qspi_nor_instance_ctrl_t
- period : agt_instance_ctrl_t , timer_cfg_t
- period_counts : timer_info_t
- periodic_ipl : rtc_cfg_t
- periodic_irq : rtc_instance_ctrl_t
- periodicIrqRateSet : rtc_api_t
- periodSet : timer_api_t
- peripheral : elc_link_t
- perm : RBLE_GATT_SET_PERM
- pga0 : adc_cfg_t , adc_instance_ctrl_t
- pga1 : adc_cfg_t , adc_instance_ctrl_t
- pga2 : adc_cfg_t , adc_instance_ctrl_t
- pga_available : adc_instance_ctrl_t

- phy_mode : sf_wifi_qca4010_status_t
- pin : ioport_pin_cfg_t , sf_touch_panel_chip_ft5x06_instance_ctrl_t , sf_touch_panel_chip_on_ft5x06_cfg_t , sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t
- pin_cfg : ioport_pin_cfg_t
- pin_output : comparator_cfg_t
- pin_reset : sf_cellular_extended_cfg_t , sf_wifi_on_gt202_cfg_t , sf_wifi_qca4010_extended_cfg_t
- pin_select : agt_input_capture_extend_t
- pin_slave_select : sf_wifi_on_gt202_cfg_t
- pinCfg : ioport_api_t
- pinDirectionSet : ioport_api_t
- pinEthernetModeCfg : ioport_api_t
- pinEventInputRead : ioport_api_t
- pinEventOutputWrite : ioport_api_t
- ping : sf_cellular_socket_api_t , sf_wifi_qca4010_onchip_stack_api_t
- pinRead : ioport_api_t
- pinsCfg : ioport_api_t
- pint_irq : ptpedmac_instance_ctrl_t
- pinWrite : ioport_api_t
- pixel_format : jpeg_decode_cfg_t , jpeg_decode_instance_ctrl_t
- pixelFormatGet : jpeg_decode_api_t , sf_jpeg_decode_api_t
- play : sf_audio_playback_hw_api_t
- playing : sf_audio_playback_common_instance_ctrl_t
- pll_cfg : cgc_clocks_cfg_t
- pll_div_max : bsp_feature_cgc_t
- pll_mul_max : bsp_feature_cgc_t
- pll_mul_min : bsp_feature_cgc_t
- pll_src_configurable : bsp_feature_cgc_t
- pll_state : cgc_clocks_cfg_t
- pllccr_type : bsp_feature_cgc_t
- pointer_hdl : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_LIST
- polarity : sdadc_channel_cfg_t
- polynomial : crc_cfg_t , crc_instance_ctrl_t
- portDirectionSet : ioport_api_t
- portEventInputRead : ioport_api_t
- portEventOutputWrite : ioport_api_t
- portRead : ioport_api_t
- portWrite : ioport_api_t
- post : sf_message_api_t
- power_lvl : sf_ble_set_tx_pwr_info_t
- power_supply_state : lpmv2_mcu_cfg_t
- pp_curr_bus_ctrl : sf_i2c_bus_t
- pp_curr_ctrl : sf_i2c_bus_t , sf_spi_bus_t
- pp_subscriber_group : sf_message_subscriber_list_t
- pp_subscriber_lists : sf_message_cfg_t , sf_message_instance_ctrl_t
- ppp_stack_size : sf_cellular_nsal_cfg_t
- preamble : sf_wifi_cfg_t
- preamble_length : phy_record_t
- pref_ops : sf_cellular_cfg_t
- press_val1 : sf_ble_blp_meas_info_t
- press_val2 : sf_ble_blp_meas_info_t
- press_val3 : sf_ble_blp_meas_info_t
- prevbuf : trng_ctrl_t
- pri_ref : ctsu_mutual_buf_t

- pri_sen : ctsu_mutual_buf_t
- priority : sf_audio_playback_common_cfg_t , sf_cellular_nsal_cfg_t , sf_message_post_cfg_t , sf_thread_monitor_cfg_t , sf_touch_panel_v2_cfg_t
- priority_group_a : adc_channel_cfg_t
- productFeatureGet : fmi_api_t
- productId : sf_ble_prf_dis_pnpid_t
- productInfoGet : fmi_api_t
- productVersion : sf_ble_prf_dis_pnpid_t
- profiling_mode_check : sf_thread_monitor_instance_ctrl_t
- profiling_mode_enabled : sf_thread_monitor_cfg_t , sf_thread_monitor_instance_ctrl_t
- prompt : sf_console_api_t
- prop : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_LIST
- property : sf_ble_char_discovery_rsp_t
- protocol_mode_val : sf_ble_prf_value_t
- prov_info : sf_cellular_instance_cfg_t , sf_wifi_qca4010_instance_cfg_t
- provisionGet : sf_ble_api_t
- provisioningGet : sf_cellular_api_t , sf_wifi_api_t
- provisioningSet : sf_cellular_api_t , sf_wifi_api_t , sf_wifi_qca4010_api_t
- provisionSet : sf_ble_api_t
- pulse_count_value : agt_input_capture_extend_t
- pulse_period_first_edge : agt_input_capture_instance_ctrl_t

- q -

- qspi_info : sf_memory_qspi_nor_instance_ctrl_t
- quality_factor : jpeg_encode_cfg_t
- queue : sf_comms_telnet_instance_ctrl_t , sf_uart_comms_instance_ctrl_t
- queue_mem : sf_uart_comms_instance_ctrl_t
- queue_size : sf_cellular_circular_queue_cfg_t , sf_wifi_qca4010_queue_cfg_t

- r -

- r : display_brightness_t , display_color_t , display_contrast_t , display_gamma_correction_t
- ram_buffer_size : sf_block_media_ram_instance_ctrl_t
- rand_num : sf_ble_sec_info_t , sf_ble_sm_enc_info_t , sf_ble_sm_key_ind_t
- randomNumberGenerate : sf_crypto_trng_api_t
- rate : i2c_cfg_t , sf_ble_blp_meas_info_t
- rd_index : ctsu_instance_ctrl_t
- read : adc_api_t , cac_api_t , can_api_t , flash_api_t , i2c_api_master_t , i2s_api_t , ptpedmac_api_t , qspi_api_t , riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t , sdmmc_api_t , sf_block_media_api_t , sf_comms_api_t , sf_console_api_t , sf_i2c_api_t , sf_memory_api_t , sf_spi_api_t , spi_api_t , trng_api_t , uart_api_t
- read32 : adc_api_t
- read_bytes_max : uart_info_t
- read_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- readIo : sdmmc_api_t

- readIoExt : sdmmc_api_t
- ready : sdmmc_info_t
- reason : sf_ble_disconnect_t
- ref : ctsu_self_buf_t
- ref_time : sf_ble_prf_tip_write_data_t
- reference_clock : cac_instance_ctrl_t
- reference_voltage : bsp_feature_adc_t
- refresh : wdt_api_t
- reg_id : adc_sample_state_t
- reg_status : sf_cellular_network_status_t
- region_info : sf_memory_qspi_nor_instance_ctrl_t
- remain : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- remote_ip : sf_cellular_socket_info_t
- remote_port : sf_cellular_socket_info_t
- rendering_enable : sf_el_gx_instance_ctrl_t
- repeat_area : transfer_info_t
- repetition : agt_input_capture_instance_ctrl_t , gpt_input_capture_instance_ctrl_t , input_capture_cfg_t
- report : sf_ble_prf_hid_report_ind_t
- report_type : sf_ble_prf_hid_report_desc_t
- req_high_throughput : sf_wifi_cfg_t
- req_type : RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_WRITE_CHAR_REQ
- reserved : sf_ble_blp_meas_info_t , sf_ble_connect_info_t , sf_ble_long_attr_info_t , sf_ble_prf_cts_curr_time_t , sf_ble_prf_cts_date_time_t , sf_ble_prf_hid_report_ind_t , sf_ble_prf_htp_temp_info_t , sf_ble_prf_ndcs_time_dst_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
- reserved2 : sf_ble_connect_info_t
- reserved3 : sf_ble_connect_info_t
- reset : cac_api_t , flash_api_t , i2c_api_master_t , sf_cellular_api_t , sf_i2c_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , timer_api_t , transfer_api_t
- reset_control : wdt_cfg_t
- reset_level : sf_cellular_extended_cfg_t , sf_wifi_qca4010_extended_cfg_t
- resolution : adc_cfg_t , sdadc_instance_ctrl_t
- resource_lock : glcd_ctrl_t
- resource_lock_tx_rx : riic_instance_ctrl_t , sci_spi_instance_ctrl_t
- resp : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t
- resp_buffer_length : sf_wifi_qca4010_instance_cfg_t
- resp_wait_time : sf_cellular_at_cmd_set_t , sf_wifi_qca4010_at_cmd_set_t
- response : sf_ble_gatt_attr_event_t
- response_required : sf_ble_write_cmd_event_data_t
- restart : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t
- restarted : riic_instance_ctrl_t , sci_i2c_instance_ctrl_t , sf_i2c_instance_ctrl_t
- resume : sf_audio_playback_api_t
- retry : sf_cellular_at_cmd_set_t
- retry_count : sf_cellular_command_parameters_info_t
- retry_delay : sf_cellular_at_cmd_set_t , sf_cellular_command_parameters_info_t , sf_wifi_qca4010_at_cmd_set_t
- riic_fastplus_rise_time : bsp_feature_riic_t
- riic_std_fast_rise_time : bsp_feature_riic_t
- ringer_cp : sf_ble_prf_ringer_cp_change_t
- ringer_setting : sf_ble_prf_ringer_setting_ntf_t
- rkey_dist : sf_ble_bonding_req_ind_t , sf_ble_bonding_start_t
- rkeys : sf_ble_bonding_response_t
- rotation_angle : sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t

- rr_interval : sf_ble_hrp_api_hrmeas_t
- rr_interval_num : sf_ble_hrp_api_hrmeas_t
- rs485_de_pin : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- rspi_clksyn : spi_on_rspi_cfg_t
- rspi_comm : spi_on_rspi_cfg_t
- rssi : sf_ble_info_t , sf_ble_scan_t , sf_cellular_info_t , sf_cellular_network_status_t , sf_wifi_info_t , sf_wifi_scan_t
- rts : sf_wifi_cfg_t
- runApply : sf_power_profiles_v2_api_t
- rx_bd_space : NX_REC
- rx_bytes : sf_cellular_stats_t
- rx_bytes_count : sci_uart_instance_ctrl_t
- rx_dest_bytes : ssi_instance_ctrl_t
- rx_dst_bytes : sci_uart_instance_ctrl_t
- rx_edge_start : uart_on_sci_cfg_t
- rx_err : sf_cellular_stats_t
- rx_fifo_trigger : uart_on_sci_cfg_t
- rx_in_use : ssi_instance_ctrl_t
- rx_pkts : sf_wifi_stats_t
- rx_thread_priority : sf_cellular_comms_extend_cfg_t
- rx_transfer_in_progress : sci_uart_instance_ctrl_t
- rx_zero_copy : sf_wifi_nsal_cfg_t
- rxi_ipl : i2c_cfg_t , i2s_cfg_t , spi_cfg_t , uart_cfg_t
- rxi_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , ssi_instance_ctrl_t

- s -

- s_addr : in_addr
- sample_clock_divisor : lvd_extend_t
- sample_count : sf_adc_periodic_cfg_t , sf_adc_periodic_instance_ctrl_t , sf_audio_record_adc_instance_ctrl_t , sf_audio_record_cfg_t
- sample_hold_mask : adc_channel_cfg_t
- sample_hold_states : adc_channel_cfg_t
- samples : sf_audio_playback_common_instance_ctrl_t
- samples_remaining : sf_audio_playback_instance_ctrl_t
- samples_total : sf_audio_playback_instance_ctrl_t
- sampleStateCountSet : adc_api_t
- sampling_freq_hz : i2s_cfg_t , i2s_info_t , ssi_instance_ctrl_t
- sampling_rate_hz : sf_audio_record_cfg_t
- SAR : dtc_reg_t
- scalarMultiplication : ecc_api_t
- scale_bits_max : sf_audio_playback_data_type_t
- scan : sf_ble_api_t , sf_wifi_api_t , sf_wifi_qca4010_api_t
- scan_cfg_mask : sdadc_instance_ctrl_t
- scan_end_b_ipl : adc_cfg_t
- scan_end_b_irq : adc_instance_ctrl_t
- scan_end_ipl : adc_cfg_t
- scan_end_irq : adc_instance_ctrl_t , sdadc_instance_ctrl_t
- scan_interval : sf_ble_cfg_t
- scan_interval_window_val : sf_ble_scps_scan_intv_change_t

- scan_mask : adc_channel_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t
- scan_mask_group_b : adc_channel_cfg_t
- scan_mode : sf_ble_scan_info_t
- scan_response_data : sf_ble_adv_info_t , sf_ble_scan_response_data_t
- scan_response_data_length : sf_ble_scan_response_data_t
- scan_trigger : sf_adc_periodic_cfg_t
- scan_window : sf_ble_cfg_t
- scanbuf : ctsu_correction_info_t
- scanCfg : adc_api_t
- scanStart : adc_api_t , ctsu_api_t , sf_touch_ctsu_api_t
- scanStatusGet : adc_api_t
- scanStop : adc_api_t
- sda_delay : i2c_cfg_t
- sdadcClockCfg : cgc_api_t
- sdadcClockDisable : cgc_api_t
- sdadcClockEnable : cgc_api_t
- sdhi_event : sdmmc_instance_ctrl_t
- sdhi_rca : sdmmc_info_t
- sdio : sdmmc_info_t
- sdio_ipl : sdmmc_cfg_t
- sdpa : ctsu_element_cfg_t
- sdramClockOutDisable : cgc_api_t
- sdramClockOutEnable : cgc_api_t
- sec : sf_ble_prf_cts_date_time_t
- sec_info : sf_ble_provisioning_t
- sec_match : rtc_alarm_time_t
- sec_mode : sf_ble_sec_info_t
- second_coefficient : ctsu_correction_info_t
- second_val : ctsu_correction_info_t
- sector_count : sdmmc_info_t
- sector_size : sdmmc_info_t
- sectorErase : qspi_api_t
- security : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_scan_t
- semaphore : sf_crypto_instance_ctrl_t , sf_el_gx_instance_ctrl_t , sf_external_irq_instance_ctrl_t , sf_message_buffer_ctrl_t::st_buffer_ctrl_flag , sf_touch_panel_v2_instance_ctrl_t
- sen : ctsu_self_buf_t
- sensor_min_sampling_time : bsp_feature_adc_t
- sensors_exclusive : bsp_feature_adc_t
- service_domain : sf_cellular_network_status_t
- service_handle : sf_ble_service_discovery_rsp_t
- set_bck_with_pckb : bsp_feature_cgc_t
- setdisplayArea : slcdc_api_t
- setExtPromiscuous : ptp_api_t
- setGcmTag : aes_api_t
- setGradientLimit : ptp_api_t
- setLocalClock : ptp_api_t
- setMasterPortID : ptp_api_t
- setMessageReceptionConfig : ptp_api_t
- setMINTevent : ptp_api_t
- setTimer : ptp_api_t
- setTxPower : sf_ble_api_t
- setup : sf_el_gx_api_t
- setWorst10Values : ptp_api_t

- sf_comms_rx_thread_stack_size : sf_cellular_comms_extend_cfg_t
- shortest_pwm_signal : gpt_instance_ctrl_t , timer_on_gpt_cfg_t
- sign : dsa_api_t , ecc_api_t , rsa_api_t
- signal : gpt_input_capture_extend_t
- signal_filter : agt_input_capture_extend_t , gpt_input_capture_extend_t
- signCrt : rsa_api_t
- signFinal : sf_crypto_signature_api_t
- signUpdate : sf_crypto_signature_api_t
- sim_status : sf_cellular_sim_pin_info_t
- simIDGet : sf_cellular_api_t
- simLock : sf_cellular_api_t
- simPinSet : sf_cellular_api_t
- simUnlock : sf_cellular_api_t
- sin_addr : sockaddr , sockaddr_in
- sin_family : sockaddr , sockaddr_in
- sin_port : sf_cellular_socket_info_t , sockaddr , sockaddr_in
- sin_zero : sockaddr , sockaddr_in
- sinfo : sf_touch_ctsu_instance_ctrl_t
- size : display_clut_cfg_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_RELIABLE_WRITE , sf_el_fx_media_partition_data_info_t , sf_el_lx_nor_memory_settings_t , transfer_info_t
- size_bytes : sf_audio_playback_data_t
- size_multiplier : sf_adc_periodic_instance_ctrl_t
- skip_internal_calibration : adc_on_sdadc_cfg_t
- slave : i2c_cfg_t
- slave_busy : riic_slave_instance_ctrl_t
- slave_latency : sf_ble_cfg_t
- slaveAddressSet : i2c_api_master_t
- slcdc_clock : slcdc_cfg_t
- slcdc_clock_setting : slcdc_cfg_t
- slope_microvolts : adc_info_t , adc_instance_ctrl_t
- SM : dtc_reg_t
- snd_ref : ctsu_mutual_buf_t
- snd_sen : ctsu_mutual_buf_t
- snoop_channel : crc_snoop_cfg_t
- snoop_direction : crc_snoop_cfg_t
- snoopCfg : crc_api_t
- snoopDisable : crc_api_t
- snoopEnable : crc_api_t
- snooze_cancel_sources : lpmv2_mcu_cfg_t
- snooze_end_sources : lpmv2_mcu_cfg_t
- snooze_request_source : lpmv2_mcu_cfg_t
- snum : ctsu_element_cfg_t
- so : ctsu_element_cfg_t
- sock_type : sf_cellular_socket_info_t
- socket_create_flag : ulpgn_socket_t
- socket_queue : sf_wifi_qca4010_instance_cfg_t
- socket_recv_buff : ulpgn_socket_t
- socket_status_buffer_length : sf_cellular_instance_cfg_t
- socketConnect : sf_wifi_qca4010_socket_api_t
- socketCreate : sf_wifi_qca4010_socket_api_t
- socketDisconnect : sf_wifi_qca4010_socket_api_t

- socketRecv : sf_wifi_qca4010_socket_api_t
- sockets : sf_wifi_qca4010_socket_ctrl_t
- socketSend : sf_wifi_qca4010_socket_api_t
- socketStatusGet : sf_wifi_qca4010_socket_api_t
- sodrv_mask : bsp_feature_cgc_t
- sodrv_shift : bsp_feature_cgc_t
- softwareEventGenerate : elc_api_t
- source_clock : cgc_clock_cfg_t
- spi_mode : qspi_instance_ctrl_t
- src_addr_mode : transfer_info_t
- ssdiv : ctsu_element_cfg_t
- ssid : sf_wifi_provisioning_t , sf_wifi_qca4010_provisioning_t , sf_wifi_qca4010_scan_t , sf_wifi_qca4010_status_t , sf_wifi_scan_t
- ssid_broadcast : sf_wifi_cfg_t
- ssl_level_keep : spi_on_rspi_cfg_t
- ssl_neg_delay : spi_on_rspi_cfg_t
- ssl_polarity : spi_on_rspi_cfg_t
- ssl_select : spi_on_rspi_cfg_t
- stage_1_gain : sdadc_channel_cfg_t
- stage_2_gain : sdadc_channel_cfg_t
- stage_num : rsa_ctrl_t
- stamp : sf_ble_blp_meas_info_t , sf_ble_prf_cts_curr_time_t , sf_ble_prf_htp_temp_info_t , sf_ble_prf_ndcs_time_dst_t
- standby_wake_sources : lpmv2_mcu_cfg_t
- start : dac_api_t , display_api_t , display_clut_cfg_t , opamp_api_t , ptp_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_message_instance_range_t , sf_touch_panel_v2_api_t , slcdc_api_t , timer_api_t , transfer_api_t
- start_bitmask : gpt_input_capture_instance_ctrl_t
- start_handle : sf_ble_char_discovery_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- start_hdl : RBLE_GATT_DISC_CHAR_DESC_REQ , RBLE_GATT_DISC_CHAR_REQ , RBLE_GATT_DISC_SVC_REQ , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , RBLE_GATT_SET_PERM , RBLE_GATT_SVC_128_LIST , RBLE_GATT_SVC_LIST , RBLE_GATT_SVC_RANGE_LIST
- start_interrupt_enabled : riic_slave_instance_ctrl_t
- start_mode : wdt_cfg_t
- startEncryption : sf_ble_api_t
- startMeasurement : cac_api_t
- startupAreaSelect : flash_api_t
- state : arc4_ctrl_t , comparator_status_t , crypto_ctrl_t , ctsu_instance_ctrl_t , display_status_t , glcd_instance_ctrl_t , ptp_cfg_t , ptp_instance_ctrl_t , sf_ble_set_tx_pwr_info_t , sf_cellular_socket_info_t , sf_el_gx_instance_ctrl_t , sf_jpeg_decode_instance_ctrl_t , sf_message_instance_ctrl_t , sf_uart_comms_instance_ctrl_t , slcdc_instance_ctrl_t
- stateGet : pdc_api_t
- station_inactivity_timeout : sf_wifi_cfg_t
- statisticsGet : sf_cellular_api_t , sf_wifi_api_t
- status : ctsu_correction_info_t , input_capture_info_t , jpeg_decode_callback_args_t , jpeg_decode_instance_ctrl_t , jpeg_encode_callback_args_t , jpeg_encode_instance_ctrl_t , lvd_callback_args_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Cfm_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Notify_Comp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Set_Data_Complete_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Set_Perm_Complete_t , rtc_info_t

, sdmmc_instance_ctrl_t , sf_ble_connect_info_t , sf_ble_disconnect_t , sf_ble_sec_enc_start_ind_t , sf_crypto_cipher_instance_ctrl_t , sf_crypto_hash_instance_ctrl_t , sf_crypto_instance_ctrl_t , sf_crypto_key_installation_instance_ctrl_t , sf_crypto_key_instance_ctrl_t , sf_crypto_signature_instance_ctrl_t , sf_el_fx_media_ebr_info_t , sf_el_fx_media_global_open_info_t , sf_el_fx_media_mbr_info_t , sf_touch_ctsu_button_info_t , sf_audio_playback_instance_ctrl_t

- statusClear : doc_api_t , lvd_api_t , wdt_api_t
- statusGet : comparator_api_t , crypto_api_t , display_api_t , doc_api_t , flash_api_t , jpeg_decode_api_t , jpeg_encode_api_t , lvd_api_t , opamp_api_t , qspi_api_t , sf_crypto_api_t , sf_jpeg_decode_api_t , wdt_api_t
- stca_mode : ptp_instance_ctrl_t
- stca_sync_mode : ptp_cfg_t
- stop : dac_api_t , display_api_t , i2s_api_t , opamp_api_t , ptp_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_touch_panel_v2_api_t , slcdc_api_t , timer_api_t , transfer_api_t
- Stop_ActivationRequest : transfer_api_t
- stop_bitmask : gpt_input_capture_instance_ctrl_t
- stop_bits : uart_cfg_t
- stop_control : wdt_cfg_t
- stop_level : gpt_output_pin_t
- stopMeasurement : cac_api_t
- stream_end : sf_audio_playback_data_t
- subosc_state : cgc_clocks_cfg_t
- sup_timeout : sf_ble_cfg_t
- sup_to : sf_ble_connect_info_t
- supports_8_bit_mmc : bsp_feature_sdhi_t
- suppress_carry_event_callback : rtc_instance_ctrl_t
- svc_code : sb_ble_prf_ias_set_alert_t
- switches : opamp_instance_ctrl_t
- sync_edge : display_output_cfg_t
- sync_polarity : display_timing_t
- sync_width : display_timing_t
- synchronization_jump_width : can_bit_timing_cfg_t
- sys_cfg : cgc_clocks_cfg_t
- system_clock : cgc_clocks_cfg_t
- systemClockFreqGet : cgc_api_t
- systemClockGet : cgc_api_t
- systemClockSet : cgc_api_t
- systickUpdate : cgc_api_t
- SZ : dtc_reg_t

- t -

- tcon_de : glcd_cfg_t
- tcon_hsync : glcd_cfg_t
- tcon_vsync : glcd_cfg_t
- tei_ipl : i2c_cfg_t , spi_cfg_t , uart_cfg_t
- tei_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t
- temp_info : sf_ble_prf_htp_temp_info_ind_t
- temp_val : sf_ble_prf_htp_temp_info_t

- text : sf_ble_anp_api_new_alert_t
- text_size : sf_ble_anp_api_new_alert_t
- thread : sf_audio_playback_common_instance_ctrl_t , sf_thread_monitor_instance_ctrl_t , sf_touch_panel_v2_instance_ctrl_t
- thread_counters : sf_thread_monitor_instance_ctrl_t
- threadRegister : sf_thread_monitor_api_t
- threadUnregister : sf_thread_monitor_api_t
- threshold : gamma_correction_t , sf_touch_ctsu_button_cfg_t , sf_touch_ctsu_slider_cfg_t , sf_touch_ctsu_wheel_cfg_t
- time : rtc_alarm_time_t
- time_segment_1 : can_bit_timing_cfg_t
- time_segment_2 : can_bit_timing_cfg_t
- time_slice : slcdc_cfg_t
- time_source : sf_ble_cts_ref_time_t
- time_zone : sf_ble_cts_local_time_t
- timeout : sf_cellular_socket_info_t , wdt_cfg_t
- timeout_clocks : wdt_timeout_values_t
- timeout_mode : riic_extended_cfg , riic_instance_ctrl_t
- timeout_period_msec : sf_thread_monitor_instance_ctrl_t
- timeout_period_watchdog_clocks : sf_thread_monitor_instance_ctrl_t
- timeout_seconds : sf_wifi_wps_t
- timeout_ticks : sf_memory_qspi_nor_cfg_t , sf_memory_qspi_nor_instance_ctrl_t
- timeoutGet : wdt_api_t
- timer_channel : ptp_callback_args_t
- tk_info : sf_ble_sm_tk_ind_t
- tk_key : sf_ble_sm_tk_ind_t
- tk_req_status : sf_ble_sm_tk_ind_t
- total : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , sci_i2c_instance_ctrl_t
- total_count : sf_el_fx_media_partition_info_t
- total_cyc : display_timing_t
- total_lines_decoded : jpeg_decode_instance_ctrl_t
- total_partitions : sf_el_fx_config_t
- total_scan_duration : sf_ble_scan_info_t
- total_size_bytes : qspi_info_t , qspi_instance_ctrl_t
- touchDataGet : sf_touch_panel_v2_api_t
- transaction_completed : sci_i2c_instance_ctrl_t
- transaction_count : riic_slave_instance_ctrl_t
- transfer_block_current : sdmmc_instance_ctrl_t
- transfer_block_size : sdmmc_instance_ctrl_t
- transfer_blocks : UX_DCD_SYNERGY_PAYLOAD_TRANSFER
- transfer_blocks_total : sdmmc_instance_ctrl_t
- transfer_dir : sdmmc_instance_ctrl_t
- transfer_flag : ptpedmac_instance_ctrl_t
- transfer_in_progress : pdc_instance_ctrl_t , sdmmc_info_t , sdmmc_instance_ctrl_t
- transfer_irq : sdmmc_instance_ctrl_t
- transfer_length_max : transfer_properties_t
- transfer_length_remaining : transfer_properties_t
- transfer_size : adc_info_t
- transfer_times : UX_DCD_SYNERGY_PAYLOAD_TRANSFER
- transfer_width : UX_DCD_SYNERGY_PAYLOAD_TRANSFER , UX_HCD_SYNERGY_PAYLOAD_TRANSFER
- transmit : sf_cellular_api_t , sf_wifi_api_t
- transpktsize : sf_cellular_qctlcatm1_socket_cfg_t
- trigger : adc_cfg_t , adc_instance_ctrl_t , comparator_cfg_t , dmac_instance_ctrl_t , dtc_instance_ctrl_t , external_irq_cfg_t , keymatrix_cfg_t , sdadc_instance_ctrl_t

- trigger_channel_0 : opamp_on_opamp_cfg_t
- trigger_channel_1 : opamp_on_opamp_cfg_t
- trigger_channel_2 : opamp_on_opamp_cfg_t
- trigger_channel_3 : opamp_on_opamp_cfg_t
- trigger_enabled : sdadc_instance_ctrl_t
- trigger_group_b : adc_cfg_t
- triggerSet : external_irq_api_t , keymatrix_api_t
- trim : opamp_api_t
- trim_capable : opamp_instance_ctrl_t
- trim_channel : opamp_instance_ctrl_t
- trim_input : opamp_instance_ctrl_t
- trim_state : opamp_instance_ctrl_t
- tsn_calib_available : adc_instance_ctrl_t
- tsn_calibration_available : bsp_feature_adc_t
- tsn_control_available : bsp_feature_adc_t
- tsn_ctrl_available : adc_instance_ctrl_t
- tsn_slope : bsp_feature_adc_t
- tuning : ctsu_instance_ctrl_t
- tuning_mutual_target_value : ctsu_cfg_t , ctsu_instance_ctrl_t
- tuning_self_target_value : ctsu_cfg_t , ctsu_instance_ctrl_t
- tunning_enable : ctsu_cfg_t
- tx_bytes : sf_cellular_stats_t
- tx_err : sf_cellular_stats_t , sf_wifi_stats_t
- tx_in_use : ssi_instance_ctrl_t
- tx_pkts : sf_wifi_stats_t
- tx_power : sf_wifi_cfg_t
- tx_src_bytes : sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- tx_timeout : sf_cellular_socket_info_t
- tx_zero_copy : sf_wifi_nsal_cfg_t
- txi_ipl : i2c_cfg_t , i2s_cfg_t , spi_cfg_t , uart_cfg_t
- txi_irq : riic_instance_ctrl_t , riic_slave_instance_ctrl_t , rspi_instance_ctrl_t , sci_i2c_instance_ctrl_t , sci_spi_instance_ctrl_t , sci_uart_instance_ctrl_t , ssi_instance_ctrl_t
- type : can_frame_t , RBLE_GATT_EVENT , sf_audio_playback_data_t , sf_ble_prf_htp_temp_info_t
- tz_upd_mode : sf_cellular_cfg_t

- u -

- uart_comm_mode : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- uart_rs485_mode : sci_uart_instance_ctrl_t , uart_on_sci_cfg_t
- underflow_1_ipl : display_cfg_t
- underflow_2_ipl : display_cfg_t
- uniqueIdGet : fmi_api_t
- unit : adc_callback_args_t , adc_cfg_t , adc_instance_ctrl_t , sdadc_instance_ctrl_t , timer_cfg_t
- unlock : sf_comms_api_t , sf_crypto_api_t , sf_i2c_api_t , sf_spi_api_t
- unread_count : sf_ble_anp_api_unread_alert_t
- unused : analog_connect_cfg_t
- update_bd_addr : sf_ble_cfg_t
- update_hz : sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- update_result : sf_ble_prf_rtus_time_updt_state_t

- update_state : sf_ble_prf_tip_write_data_t
- updateAnnounceFlags : ptp_api_t
- updateAnnounceMsgs : ptp_api_t
- updateClockID : ptp_api_t
- updateDelayMsgInterval : ptp_api_t
- updateDomainNumber : ptp_api_t
- updateFlashClockFreq : flash_api_t
- updateHash : hash_api_t
- updateSyncAnnounceMsgInterval : ptp_api_t
- usbClockCfg : cgc_api_t
- username : sf_cellular_provisioning_t
- uuid : RBLE_GATT_CHAR_128_LIST , RBLE_GATT_CHAR_DESC_128_LIST , RBLE_GATT_CHAR_LIST , RBLE_GATT_INCL_128_LIST , RBLE_GATT_INCL_LIST , RBLE_GATT_READ_CHAR_REQ , sf_ble_char_desc_discovery_rsp_t , sf_ble_char_discovery_req_t , sf_ble_char_discovery_rsp_t , sf_ble_char_read_req_t , sf_ble_service_discovery_req_t , sf_ble_service_discovery_rsp_t
- uuid128 : sf_ble_uuid_t
- uuid16 : sf_ble_uuid_t
- uuid32 : sf_ble_uuid_t
- uuid_length : sf_ble_uuid_t
- ux_dcd_synergy_D0_fifo_state : UX_DCD_SYNERGY
- ux_dcd_synergy_D1_fifo_state : UX_DCD_SYNERGY
- ux_dcd_synergy_ep_slave_transfer_request_semaphore : UX_DCD_SYNERGY_ED
- ux_dcd_synergy_pipe : UX_DCD_SYNERGY
- ux_synergy_next_available_bufnum : UX_HCD_SYNERGY

- v -

- val_hdl : RBLE_GATT_SET_DATA
- val_len : RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Mult_Resp_t , RBLE_GATT_SET_DATA , RBLE_GATT_WRITE_CHAR_REQ , sf_ble_long_attr_info_t
- valid_opamps : opamp_instance_ctrl_t
- value : RBLE_GATT_DESIRED_TYPE , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Ind_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Handle_Value_Notif_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Desc_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Read_Char_Long_Resp_t , RBLE_GATT_EVENT::Event_Gatt_Parameter_u::RBLE_GATT_Write_Cmd_Ind_t , RBLE_GATT_QUERY_RESULT , RBLE_GATT_RELIABLE_WRITE , RBLE_GATT_SET_DATA , RBLE_GATT_UUID_TYPE , RBLE_GATT_WRITE_CHAR_REQ , RBLE_GATT_WRITE_RELIABLE_REQ , sf_ble_long_attr_info_t , sf_ble_prf_hid_report_desc_t
- value_handle : sf_ble_char_discovery_rsp_t
- value_size : RBLE_GATT_DESIRED_TYPE , RBLE_GATT_UUID_TYPE , sf_ble_prf_hid_report_desc_t
- variant : gpt_input_capture_instance_ctrl_t , gpt_instance_ctrl_t , input_capture_info_t
- vendorId : sf_ble_prf_dis_pnpid_t
- vendorIdSource : sf_ble_prf_dis_pnpid_t
- verify : dsa_api_t , ecc_api_t , rsa_api_t

- verifyFinal : sf_crypto_signature_api_t
- verifyUpdate : sf_crypto_signature_api_t
- version : sf_ble_chipset_info_t , sf_i2c_api_t , sf_spi_api_t , sf_wifi_ip_addr_t , sf_wifi_qca4010_ip_addr_t
- version_id : ssp_pack_version_t , ssp_version_t
- versionGet : adc_api_t , aes_api_t , analog_connect_api_t , arc4_api_t , cac_api_t , can_api_t , cgc_api_t , comparator_api_t , crc_api_t , crypto_api_t , ctsu_api_t , dac_api_t , display_api_t , doc_api_t , dsa_api_t , ecc_api_t , elc_api_t , external_irq_api_t , flash_api_t , fmi_api_t , hash_api_t , i2c_api_master_t , i2c_api_slave_t , i2s_api_t , input_capture_api_t , ioport_api_t , jpeg_decode_api_t , jpeg_encode_api_t , key_installation_api_t , keymatrix_api_t , lpmv2_api_t , lvd_api_t , opamp_api_t , pdc_api_t , ptp_api_t , ptpedmac_api_t , qspi_api_t , rsa_api_t , rtc_api_t , sdmmc_api_t , sf_adc_periodic_api_t , sf_audio_playback_api_t , sf_audio_playback_hw_api_t , sf_audio_record_api_t , sf_ble_api_t , sf_ble_onboard_profile_api_t , sf_block_media_api_t , sf_cellular_api_t , sf_cellular_socket_api_t , sf_comms_api_t , sf_console_api_t , sf_crypto_api_t , sf_crypto_cipher_api_t , sf_crypto_hash_api_t , sf_crypto_key_api_t , sf_crypto_key_installation_api_t , sf_crypto_signature_api_t , sf_crypto_trng_api_t , sf_el_gx_api_t , sf_external_irq_api_t , sf_jpeg_decode_api_t , sf_memory_api_t , sf_message_api_t , sf_power_profiles_v2_api_t , sf_socket_api_t , sf_thread_monitor_api_t , sf_touch_ctsu_api_t , sf_touch_panel_chip_api_t , sf_touch_panel_v2_api_t , sf_wifi_api_t , sf_wifi_onchip_stack_api_t , sf_wifi_qca4010_api_t , sf_wifi_qca4010_onchip_stack_api_t , sf_wifi_qca4010_socket_api_t , slcdc_api_t , spi_api_t , tdes_api_t , timer_api_t , transfer_api_t , trng_api_t , uart_api_t , wdt_api_t
- vertical_resolution : jpeg_encode_cfg_t , jpeg_encode_instance_ctrl_t , jpeg_encode_raw_image_parameters
- voltage_ref : adc_cfg_t , adc_instance_ctrl_t
- voltage_slope : lvd_cfg_t
- voltage_threshold : lvd_cfg_t
- volume : sf_audio_playback_common_instance_ctrl_t
- volumeSet : sf_audio_playback_api_t
- vref_src : adc_on_sdadc_cfg_t
- vref_voltage : adc_on_sdadc_cfg_t
- vsize : display_input_cfg_t , glcd_ctrl_t
- vsize_pixels : sf_touch_panel_chip_on_sx8654_cfg_t , sf_touch_panel_chip_sx8654_instance_ctrl_t , sf_touch_panel_v2_cfg_t , sf_touch_panel_v2_instance_ctrl_t
- vsync : pdc_state_t
- vsync_polarity : pdc_cfg_t , pdc_instance_ctrl_t
- vtiming : display_output_cfg_t

- w -

- wait : sf_external_irq_api_t , sf_jpeg_decode_api_t
- wait_option : sf_crypto_cfg_t , sf_crypto_instance_ctrl_t
- wave_form : slcdc_cfg_t
- wds : sf_wifi_cfg_t
- wdt_open : wdt_instance_ctrl_t
- whitelistAdd : sf_ble_api_t
- whitelistDel : sf_ble_api_t
- wifiStatusGet : sf_wifi_qca4010_api_t
- window_end : wdt_cfg_t

- window_start : wdt_cfg_t
- winfo : sf_touch_ctsu_instance_ctrl_t
- wmm : sf_wifi_cfg_t
- word_length : i2s_cfg_t
- work_buffer : aes_ctrl_t
- work_memory_size_bytes : sf_message_cfg_t
- wps_key : sf_wifi_wps_t
- wps_mode : sf_wifi_wps_t
- wpsStart : sf_wifi_api_t
- wr_index : ctsu_instance_ctrl_t
- wr_offset : RBLE_GATT_WRITE_CHAR_REQ
- write : can_api_t , dac_api_t , doc_api_t , flash_api_t , i2c_api_master_t , i2s_api_t , sdmmc_api_t , sf_block_media_api_t , sf_comms_api_t , sf_console_api_t , sf_i2c_api_t , sf_memory_api_t , sf_spi_api_t , slcdc_api_t , spi_api_t , uart_api_t
- write_bytes_max : uart_info_t
- write_irq : ctsu_cfg_t , ctsu_instance_ctrl_t
- write_protect : sdmmc_extended_cfg_t , sdmmc_instance_ctrl_t
- write_protected : sdmmc_info_t
- writeIo : sdmmc_api_t
- writeIoExt : sdmmc_api_t
- writeRead : i2s_api_t , sf_spi_api_t , spi_api_t
- ws_continue : i2s_cfg_t

- x -

- x : display_coordinate_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_payload_t
- x_capture_pixels : pdc_cfg_t , pdc_instance_ctrl_t
- x_capture_start_pixel : pdc_cfg_t , pdc_instance_ctrl_t
- x_resolution_pixels : pdc_instance_ctrl_t
- xip_mode : qspi_instance_ctrl_t

- y -

- y : display_coordinate_t , sf_touch_panel_v2_calibrate_t , sf_touch_panel_v2_payload_t
- y_capture_pixels : pdc_cfg_t , pdc_instance_ctrl_t
- y_capture_start_pixel : pdc_cfg_t , pdc_instance_ctrl_t
- y_resolution_pixels : pdc_instance_ctrl_t
- year : sf_ble_prf_cts_date_time_t
- year_match : rtc_alarm_time_t

- z -

- zeroPaddingDecrypt : aes_api_t

- zeroPaddingEncrypt : aes_api_t
- zeros_written : ssi_instance_ctrl_t

Renesas Synergy™ SSP

User’s Manual

Publication Date: Revision 1.70 Mar.18.2025

Renesas Synergy™ SSP V2.7.0

User's Manual

Renesas Electronics Corporation